NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

LEVEL OF DETAIL MODELS FOR DISMOUNTED INFANTRY IN NPSNET-IV.8.1

by

Christopher Allen Chrislip
James Frederick Ehlert, Jr.

September 1995

Thesis Advisor: David R. Pratt
Co-Advisors : Michael J. Zyda
Shirley M. Pratt

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
LEVEL OF DETAIL MODELS FOR DISMOUNTED INFANTRY IN NPSNET-IV.8.1(U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Chrislip, Christopher Allen
Ehlert, James Frederick, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

NPSNET-IV.7J has a limited capability to display up to 10 Dismounted Infantry (DI) icons due to the enormous number of rendered polygons and computational load required. In order to provide a more realistic training scenario for the user, NPSNET-IV.8.1 requires the capability to display at least 150 independent DI entities in real-time.

Requirements were satisfied by creating low-level and medium level-of-detail autonomous and controlled DI icons and their associated motion algorithms to support existing DI models in NPSNET-IV.8.1. Additionally, view volume culling techniques were employed to reduce polygon flow through the graphics pipeline and enhance system speed.

This research enhances the capabilities of the NPSNET-IV.8.1 to be able to display 150 DI icons.The inclusion of level of detail models composed of fewer polygons than an existing high level model, together with view volume culling ensure NPSNET-IV.8.1 can satisfy sponsor requirements of displaying 150 DI icons and still maintain a frame rate of 10-15 frames per second.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Level of Detail Models, Dismounted Infantry, View Volume Culling	98
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

LEVEL OF DETAIL MODELS FOR DISMOUNTED INFANTRY IN NPSNET-IV.8.1

Christopher Allen Chrislip
Lieutenant, United States Navy
B.S.N.E., North Carolina State University, 1990

and

James Frederick Ehlert, Jr.
Lieutenant, United States Navy
B.S.M.E., U.S. Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Authors: Christopher Allen Chrislip James Frederick Ehlert, Jr.

Approved by:

David R. Pratt, Thesis Advisor

Michael J. Zyda, Co-Advisor Shirley M. Pratt, Co-Advisor

Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

NPSNET-IV.7J has a limited capability to display up to 10 Dismounted Infantry (DI) icons due to the enormous number of rendered polygons and computational load required. In order to provide a more realistic training scenario for the user, NPSNET-IV.8.1 requires the capability to display at least 150 independent DI entities in real-time.

Requirements were satisfied by creating low-level and medium level-of-detail autonomous and controlled DI icons and their associated motion algorithms to support existing DI models in NPSNET-IV.8.1. Additionally, view volume culling techniques were employed to reduce polygon flow through the graphics pipeline and enhance system speed.

This research enhances the capabilities of the NPSNET-IV.8.1 to be able to display 150 DI icons.The inclusion of level of detail models composed of fewer polygons than an existing high level model, together with view volume culling ensure NPSNET-IV.8.1 can satisfy sponsor requirements of displaying 150 DI icons and still maintain a frame rate of 10-15 frames per second.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

There are several people who deserve a tremendous amount of thanks and credit for their support during the development of this work. First, we wish to thank Dr. David R. Pratt, our thesis advisor and primary motivator. Without his support and direction, we would have bumbled down the path of Artificial Intelligence. Thanks for the help and motivation Dave! Enjoy the Diet Coke. Secondly, we wish to thank Dr. Michael Zyda, our thesis co-advisor, whose kept us motivated throughout all of our little failures and accomplishments along the way.

We especially wish to thank our most important co-advisor and co-programmer, Shirley M. Pratt who literally made this all possible. From day one, Shirley taught us to turn on NPSNET, draw some Dudes, and make them walk. Without Shirley's day-to-day support, guidance, de-bugging skills and patience, and her undying perseverance to help us make this work, we would still be just flying helicopters. We can not thank you enough Shirley!

Finally, we wish to thank our families and friends for their help and support to see us through this research. Thanks for standing by us while we kept our heads in the computers.

I. INTRODUCTION

A. BACKGROUND

The United States military is committed to the development of virtual world technologies and distributed interactive simulation (DIS) to more effectively and economically train warriors of the future [HLMOHN]. Effective implementation of DIS may potentially remove the necessity of expensive training travel and large scale exercises. All personnel involved in computerized simulations will benefit from their ease of use and real-time feed back in training scenarios. DIS has proven itself effective in providing networked interaction between a limited number of participants across the nation. As requirements for more participants and more realistic activities steadily increase, DIS must be able to keep pace with the quickly changing needs of users. It must be able to handle the increase in users and the large amounts of information interchanged in the network.[HLMOHN]

The main driving force behind large-scale virtual simulations has been the United States Army. Their vision to have many thousands of independent entities participating in simulations by the end of the century drives the need to make entities appear more realistic and versatile in their interaction in a virtual world. As more entities are added to a world, its realism is increased. Unfortunately, additional players and more realistic entity icons come at the expense of system speed and operational trade-offs. Inclusion of more icons slows down the system, thereby reducing the reality one seeks to achieve. As this is an undesirable effect, many speed-up techniques are implemented to reduce this while retaining system realism and user interest. The emphasis of this joint thesis is to provide one method of rendering to enhance system realism while not reducing overall system speed.

B. PROBLEM

In order to ensure the virtual battlefield is as realistic as possible, NPSNET-IV.8.1 designers continue to develop models for the system that replicate, as closely as possible, actual features in the real world. These models include life-like trees, shrubs, highly

detailed buildings, trucks, jets, tanks, helicopters, ships, submarines, and people. NPSNET-IV.8.1 is currently successful in accurately portraying most of these entities at real-time rendering speeds. However, the system is limited to the number of entities it can accurately render at high speeds before one experiences noticeable system degradations. When networking the system on a worldwide scale, system degradation may be more pronounced, further reducing the number of entities that may participate.

The recent inclusion of the University of Pennsylvania's Jack Motion Library (Jack ML) software for rendering high resolution humans has added a new feature to NPSNET-IV.7J that was previously unavailable. While Jack ML has become the solution for rendering high resolution humans in NPSNET-IV.8.1, the Jack ML model has proven too cumbersome for required large scale exercises where many foot soldiers (dismounted infantry or DI) are needed. The Jack ML model used in NPSNET-IV.8.1 for high resolution rendering is composed of 478 individual polygons [JPGRAN]. While this model is highly efficient for rendering DI in the foreground, such a model is not efficient for entities beyond high resolution perception where a lower level of detail (LOD) model could be used. The current implementation of Jack ML does not provide for LOD models as an entity moves further from one's view frustum. As such, the 478-polygon model is rendered at all viewing ranges. As a result, creation of more than 20 DI entities on NPSNET-IV.8.1 degrades system functionality, namely frame rate, well below acceptable requirements. Since this is clearly not desired, and NPSNET-IV.8.1 sponsors require at least 150 DI entities be efficiently renderable in the near term, LOD models of the DI are required to make this possible while allowing other entities to be rendered at the same time.

C. FOCUS / APPROACH

In order to address the speed and rendering requirements of NPSNET-IV.8.1, two techniques are employed. The first is the design and implementation of three LOD DI models. The second uses view frustum culling to save rendering time.

1. LOD Models

With the sponsor required increase of DI entities, current NPSNET-IV.8.1 Jack ML models are augmented with three more distinct level of detail models. These LOD models are provided to the NPSNET-IV.8.1 system software and are automatically loaded based on user view point range from a displayed DI entity. Using the capabilities designed into IRIS Performer Application Programming Interface (API) software, LOD models are loaded as needed using the pfLOD command. The features provided by the software allow users to define specific level of detail models and ranges for rendered entities. A key aspect of the LOD models designed for DI entities is that they have significantly fewer individual polygons and articulated joints per model. This large reduction in polygon count and articulations reduces overall system degradation and allows the creation of the 150 DI entities required.

2. View Volume Culling

In order to preserve frame rendering rates where DI entities are involved, view volume culling is implemented to reduce system overhead for field of view rendering. This technique utilizes one's field of view in the view frustum to determine what is visible and what is potentially visible in the scene. Entities not potentially visible to the viewer are not considered during frame rendering, and only those entities visible or potentially visible are rendered. This principle may be further extended to objects potentially visible but not readily viewable, e.g., directly behind one's view point. This reduction in polygons per scene ensures only pertinent information is provided to the user and further reduces the number of polygons that are required to be rendered. Implementing this feature with Jack ML and the LOD models further ensures the inclusion of 150 DI entities in NPSNET-IV.8.1.

D. ORGANIZATION

A discussion of previous work related to LOD models is presented in Chapter II. It includes a description of the Jack ML model, LOD models currently in use in NPSNET-

IV.8.1 and other sites, LOD construction considerations, and how IRIS Performer API software handles this information.

Chapter III is comprised of a discussion of the methodology used and final design for the DI entity LOD models in NPSNET-IV.8.1. A description is provided for the basic construction of the models, their associated ranges used for viewing purposes.

In Chapter IV, a discussion is provided for the methodology and implementation of view volume culling as is done with the DI entity LOD models. This chapter discusses some possible alternatives for implementing this technique, and an explanation of why one method is chosen over another.

Chapter V discusses the final implementation of the Dude models in NPSNET-IV.8.1 and their associated motion algorithms required to support movement of a DI entity. This chapter also provides a concise report of the results of this research, including current NPSNET-IV.8.1 data as compared to previous data on frame rates while using the LOD models with and without view volume culling. Chapter VI concludes this research with a discussion of the results, and some insights for future work needed to further enhance the capabilities of NPSNET-IV.8.1.

II. PREVIOUS WORK

Distributed Interactive Simulation (DIS) technology over the past several years has concentrated on the interaction of vehicles. The dismounted infantryman (e.g., the individual foot soldier) has been largely ignored or was only represented as a static model. Recently NPS, SARCOS Inc., and the University of Pennsylvania, under the direction of the Army Research Laboratory, demonstrated the ability to insert a fully articulated human figure into a DIS environment [DRPRATT2]. This advancement has paved the way for NPSNET-IV.7J to fully integrate humans in the virtual world. NPSNET-IV.7J has shown great improvement in motion depiction from the previously scripted, non-distributed motion studies that normally used stick figures. Before this work was implemented, several background principles were exploited that now make human insertion in a virtual world possible. They are: a) LOD and IRIS Performer Features, b) DI_Guy Development, and c) Jack ML, all of which are discussed below.

A. IRIS PERFORMER FEATURES

NPSNET-IV relies on the IRIS Performer API for graphics rendering. IRIS Performer includes built-in functions that make frame rate manipulation possible for a given application. By taking advantage of these properties, it is possible to create an effective and realistic virtual world that is nearly unhampered by frame rate limitations [IRIS]. Some of the more applicable IRIS Performer features of interest are discussed below.

1. Level of Detail Handling and IRIS Performer Software

Level of detail models are useful because they easily exploit the principle that perspective perceptions of discernible details and size reduce with increasing distance from the observer, as is demonstrated in Figure 1. Simply stated, objects further from the viewer need not be rendered using full polygon sets. This principle allows programmers to use lower resolution models for distant objects. Since finer details are less pronounced, they need not be rendered and may be left out of the model thereby putting fewer polygons into

the graphics pipeline, allowing for higher frame rates. Important to note is that when polygons transform to less than one pixel, they effectively combine [DRPRATT1]. As distances increase, this natural occurrence aids in reducing visual detail and supports the use of LOD models. Several factors should be considered when rendering visual icons. Environmental effects such as clouds, fog, smoke, and haze will further reduce a viewer's perception of an icon and lower resolution models may be more appropriate. Moreover, for most purposes, distance is the most significant factor in determining what type of model to use. As such, distance is the primary factor used in NPSNET-IV.8.1 LOD model design. While there are several types of LOD models in use, this section concentrates on icon processing.

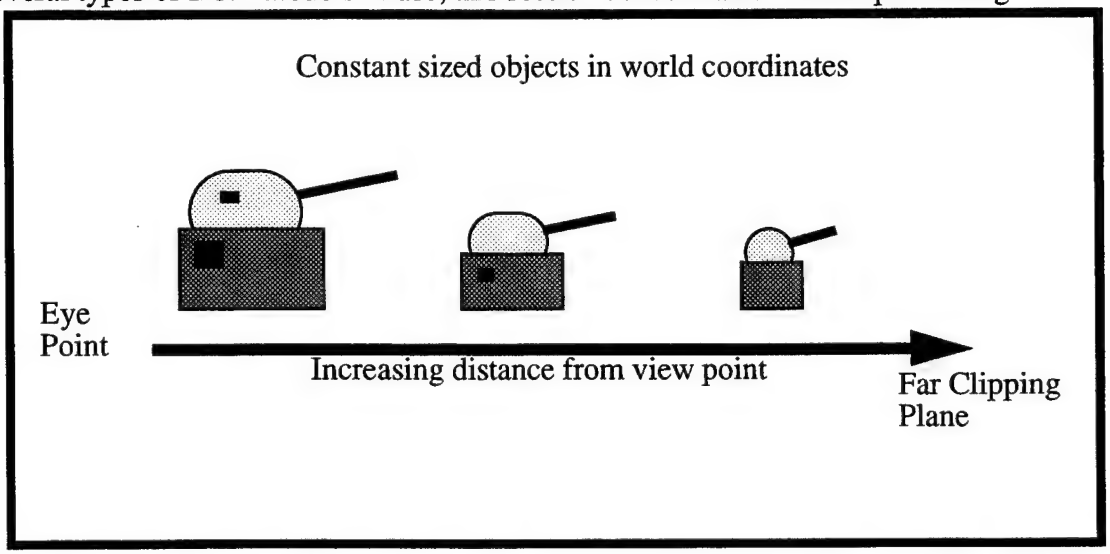


Figure 1. Perspective Projection on Apparent Size

a. Icons

Whether an object is fixed or moving, the LOD switching algorithm is the same. As shown in Figure 2, the icon rendered, i.e., high LOD, medium LOD, or low LOD,

is chosen based upon which bin the distance from the view point to the icon falls [DRPRATT1].

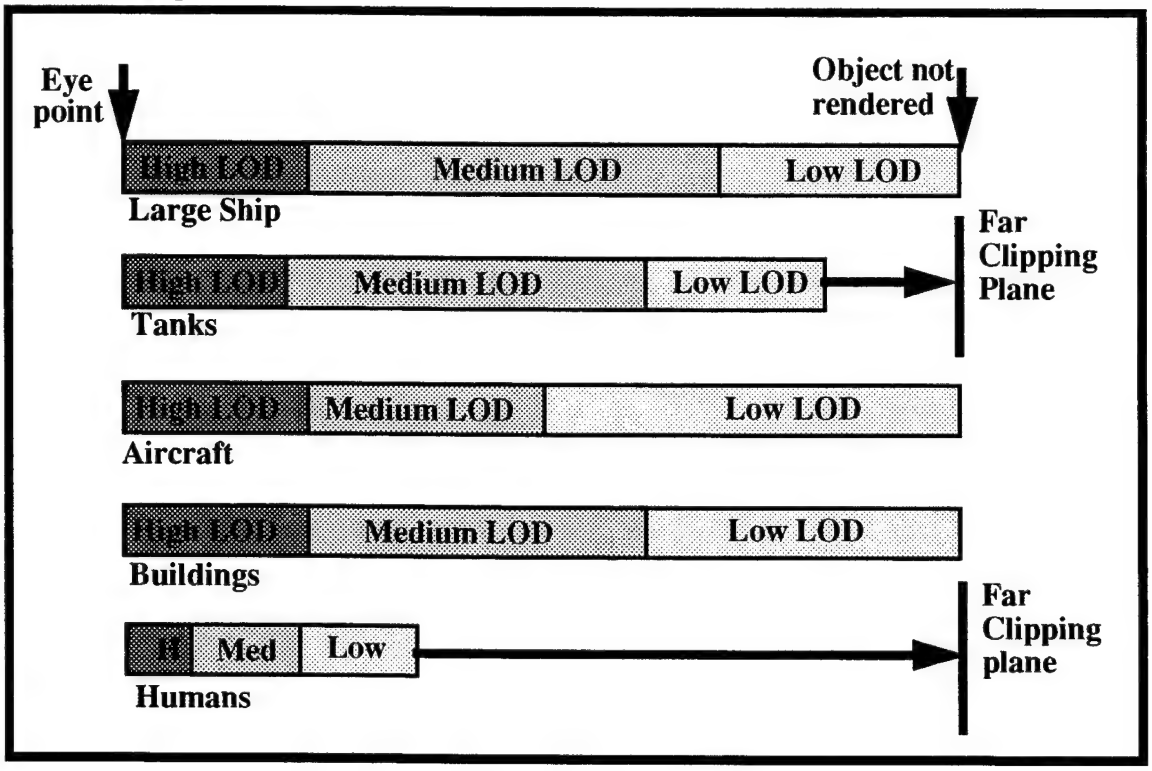


Figure 2. Level of Detail Ranges

Although this appears to be a simple switch based upon distance, there is no exact algorithm to determine where or when the switch should occur. Trial and error seems to work best. There are some rules to follow to help in this determination. Most important

is the need to preserve the outline of the icon. As demonstrated in Figure 3, an outline change can be noticeable if it is done too close to the user view point [DRPRATT1].

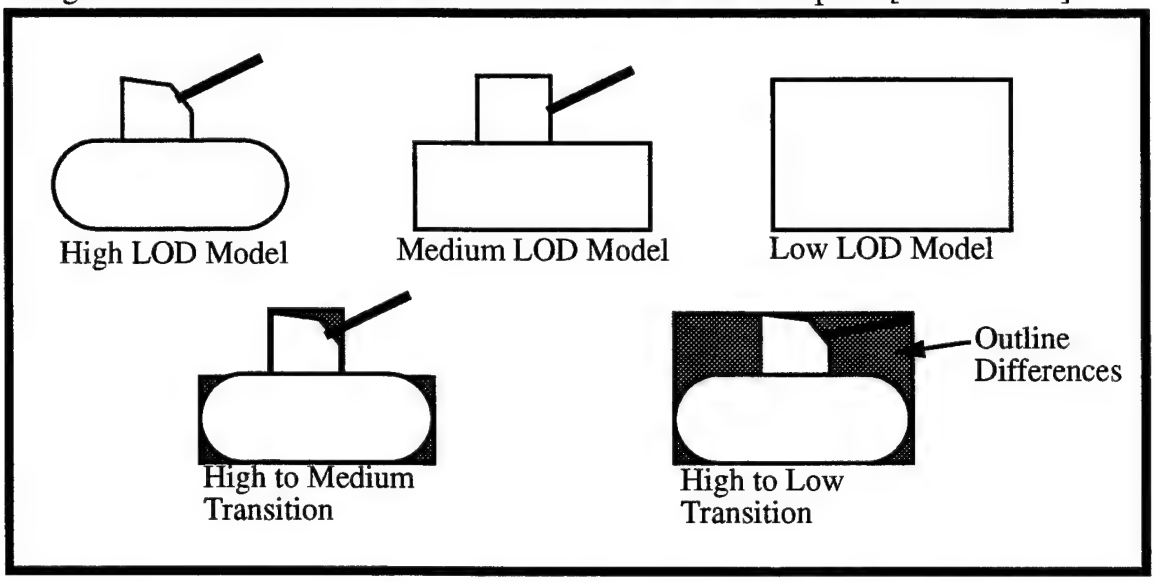


Figure 3. Effect of Differing Outlines on Model Switching

Color and texture are also important to consider when switching models. Colors and textures may not appear the same in the distant model as compared to the closer. Such changes may clearly indicate where a boundary exists. Here again, trial and error may be the most appropriate method to solve model LOD switching inconsistencies.

Another fundamental problem, outline inconsistencies, can occur when switching models. As shown in Figure 2, the point of switch between models may be a discrete snap transition from one frame to the next, and a switch between models may occur in exactly one frame. This may draw attention to the transition, detracting from the overall affect of the scene. Blending allows two LOD models to be displayed at the same time, and it slowly blends differences from one to the other. While this method, possible with IRIS Performer, allows a smooth transition from one model to the next, the displaying of both models and the blending of alpha values during the transition actually increases the system load during the transition [IRIS]. A second method, morphing, allows users to define specific points on the models and merges several polygons into one. Whether using snap

changes, blending, or morphing, all methods are guaranteed to reduce final polygon count.[DRPRATT1]

b. IRIS Performer Features

IRIS Performer software allows programmers to define specific LOD models to use for switching. By customizing such models, switching transition inconsistencies and system load problems may be reduced. Key to IRIS Performer's LOD switching control is the pfLOD command. Each child, i.e. LOD model, has a defined switching range, and the entire pfLOD has a defined center [IRIS]. Table 1 from the IRIS Performer Programming Guide shows some of the built-in Performer functions for use in LOD modeling.

Function	Description
pfNewLOD	Create a level of detail node.
pfLODRange	Set a range at which to use a specified child node.
pfGetLODRange	Find out the range for a given node.
pfLODCenter	Set the pfLOD center.
pfGetLODCenter	Retrieve the pfLOD center.

Table 1. IRIS Performer LOD Functions [IRIS]

A sophisticated scheme for smooth switching may contain multiple LOD models that are grouped under a single LOD node for a specific geometry. This technique can ensure a smooth transition for a given icon. Figure 4 shows a simple node with several children numbered 1 through *n*. LOD 1 is the highest resolution model, with resolution decreasing with increasing n. Additionally, each child may have several LOD components also defined. Associated with each node is a list of ranges that define the distance at which

each model should be displayed. There is no limit to the number of LOD models that may be used in IRIS Performer [IRIS].

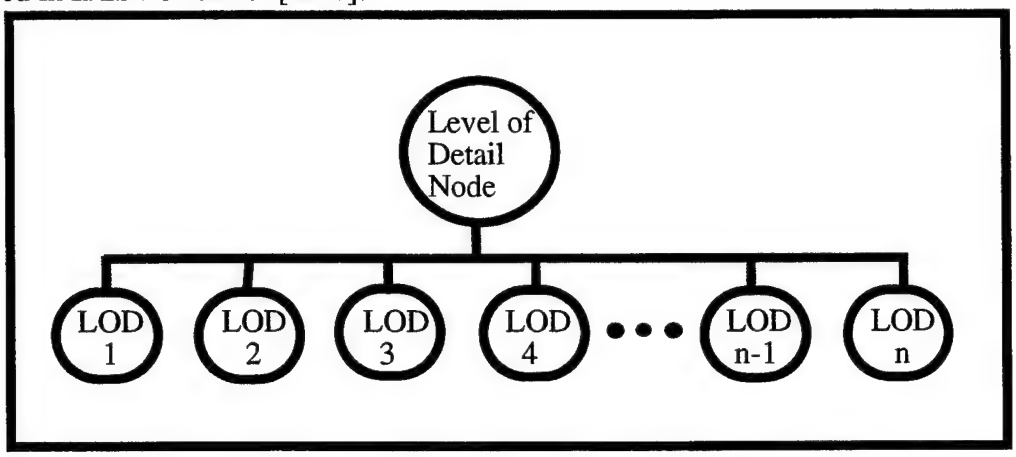


Figure 4. Level of Detail Node Structure [IRIS]

The pfLOD node contains a value known as the center of the LOD processing. The center is the x, y, z location that defines the point used in conjunction with the user's view point for range switching. Figure 5 shows an example using multiple switching models.[IRIS]

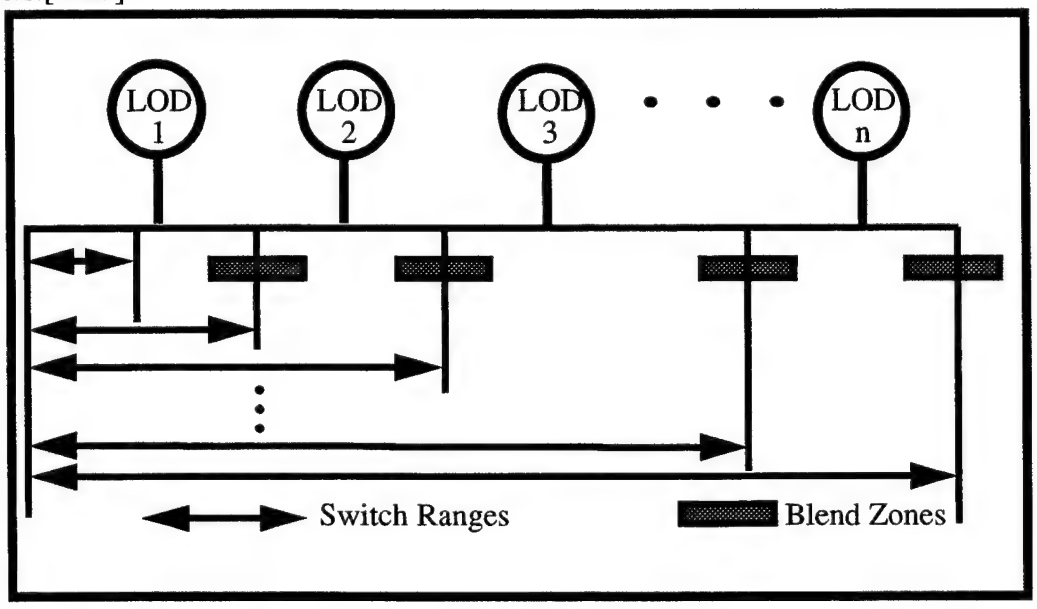


Figure 5. Level of Detail Processing [IRIS]

This example, from the IRIS Performer Programming Guide, shows a row of identical trees placed at increasing range from the view point. Double ended arrows indicate switching ranges for each level of detail. Range bracketing is used until the final range is passed, at which nothing is drawn. The pfLOD node's switch range list contains one more entry than the number of child nodes for this purpose.[IRIS]

The LOD switch ranges are processed before making the LOD selection. The goal of range setting is to switch LODs as objects reach a certain level of perceptibility. Items effecting this change are the size of the channel in pixels, field of view, and distance from the view point. Performer uses a channel size of 1024x1024 pixels and a 45-degree field of view as the basis for calculating LOD switching ranges.[IRIS] These help determine what the user will see and how complex the scene may be.

When switching between levels, model outline differences can be apparent when model discontinuities make the switch appear to change overall shape. IRIS Performer can relieve much of this by allowing users to define a blend zone where alpha values are blended for the closer model and the next model, making the transition much smoother [IRIS].

c. LOD Models in use

As stated above, many examples of LOD models can be found throughout the virtual reality programming world. NPSNET-IV.8.1 currently uses LOD models when rendering tanks, helicopters and jets. These applications have proven effective in reducing polygon count while preserving the overall entity shape. Another form used in NPSNET-IV.8.1 is the "dead" model where an icon is turned black to indicate it is inactive. Other applications of LOD icons may be found in contemporary work world wide.

An excellent example is the work of Funkhouser, et.al. of the University of California at Berkeley for the walkthrough of Soda Hall. This model is a detailed 3D model of a building complete with furniture and realistic lighting. Of note here is that while this is a building model and not a large scale virtual world, LOD models were necessary to pro-

vide high speed rendering and realistic walkthrough speed. For example, the designers used five LOD models for a desk to improve refresh rates and memory utilization. These models are adjusted so that transitions between levels are barely noticeable. The authors provide excellent descriptions of other models used in their system, and they chose to use blending techniques to achieve switching consistency. Little overhead was incurred during blends since differences in levels of complexity from one model to the next were very small.[TAF-UNK]

B. DI_GUY

In response to the now defunct Soviet threat, the first generation of low-cost, networked simulation systems, SIMNET or Simulation Network, was developed to simulate primarily company-scale tank battles. Using tank optics as the model, SIMNET was limited to an 89 degree field of view, less than half that of the 180 degree human view. This simplifying factor led to the creation of the terrain database based upon tank warfare so that only features related to this were rendered.[THORP] DIS, the successor of SIMNET is also vehicle based. As threats changed, emphasis has now shifted toward individual soldier intensive simulations, and the need for human simulation is growing.

Small regional conflicts, Special Operations, and Military Operations in Urban Terrain are not currently cost efficient and do not lend themselves well to simulation. In these simulations, small units of foot soldiers interact to accomplish a mission. Clearly, the need to simulate and train based upon small scale conflicts has grown. Profound benefits of such simulations are certain to be well worth the effort. As such, Pratt et.al., demonstrated the new human in DIS for this purpose at the 1994 Individual Combatant Modeling and Simulation Symposium, INCOMSS-94. In early SIMNET simulations, humans were represented as a texture map, with changes in posture represented by different textures. Articulations were limited or not truly existent for the humans. DIS protocols have an extensible method of rendering articulations[IDA]. It was this feature that the INCOMSS-94 demonstration team made use of to articulate its model.[DRPRATT2]

For the INCOMSS-94 project, the model used was a Jack human figure created by the University of Pennsylvania. The model was converted to MultiGen Flight format so that NPSNET-IV.7's IRIS Performer software could load the entity like other models. As seen in Figure 6, this model has 39 degrees of freedom in 17 separate joints.

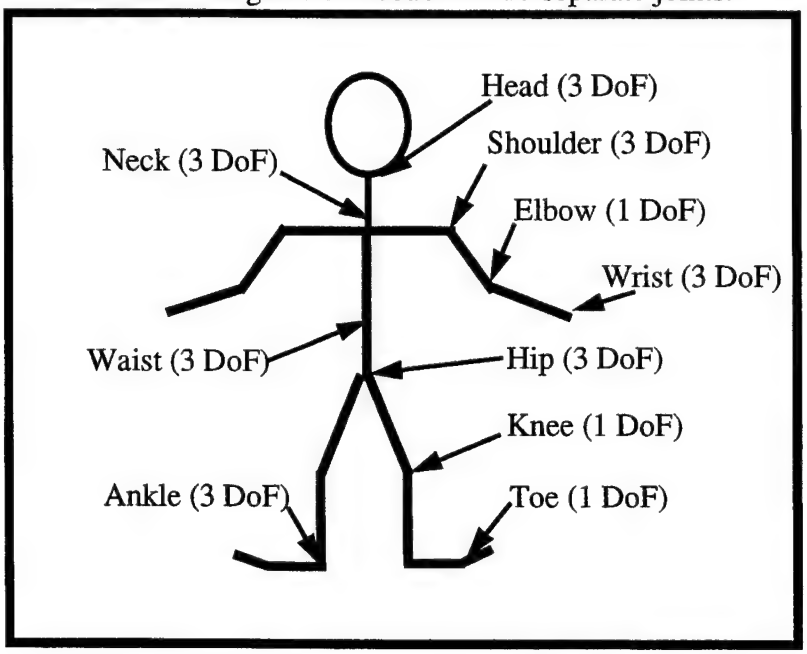


Figure 6. Schematic of Human (DI_guy)

This representation for a DI, called DI_guy, was used in conjunction with the Jack software. Each entity in NPSNET-IV is part of a class of entities. From the class Ground Vehicle is derived the subclass Person for which a subclass dismounted infantry (DI_guy) was developed. The DI_guy subclass is based on the person vehicle, simply a vehicle such as a tank or jeep with restricted speed. The DI_guy supports 16 articulations, each change of which must be broadcast to the network in DIS format as part of a PDU used for controlling ground-based entities, called the Entity-state PDU. This PDU provides the essential information needed for its units, e.g., speed, heading, etc. All input for this class comes over a custom designed network socket and communicates with the Jack process, usually running on a separate machine. Real-time raw data is manipulated by Jack. Jack manipulation

data is then sent to NPSNET-IV.7 and the DI_guy is updated on users' displays. Communications packets are sent at a rate of 30 times per second.[PTBARH]

The DI_guy process is a communications server, elevation server, and data display device. As a communications server, it dead reckons the human figure icons and formats DIS-compliant PDUs. A copy of the terrain database is maintained to provide ground elevation information and slope for a given location. One of the primary uses of the DI_guy is to debug the system by showing current location status and parameter values. The DI_guy data are updated at 60 Hz. Once packets are received, DI_guy computes elevation based upon x-y. This information can be passed to a user who is controlling the DI_guy to provide force feedback information if the DI_guy is being driven by a single user on some sort of simulation device. These data packets are then passed to Jack to compute the remaining joint angles for the DI_guy's arms and legs. Once filled in, DI_guy packets are sent to the NPSNET-IV.7 display devices for updates in rendering. [DRPRATT2]

Locomotion is based on the global velocity vector and global heading of the unit. Motion is a series of scripted sequences based on velocity. The current time is recorded at the beginning of each footstep, and the time at each update is used to determine the proper frame of the stride to display [DRPRATT2]. This calculation limits the overall speed of the DI_guy icon. All information is then broadcast to the network in DIS PDU format. Locomotion computations are only performed when the figure is in the standing posture. All other forms take a scripted roll and are sent to the network as the new posture. The posture can change only when the figure is not walking. Additionally, a mechanism is provided for forced stops, collisions, and stopping that are each handled differently and sent to the network for display.[DRPRATT2]

1. NPSNET-IV.7 Input Sources For DI_guy

Three display devices, two Head Mounted Displays and the Walk-In Synthetic Environment are used to interact as a DI_guy on NPSNET-IV.7. Since the user can see other non-Individual Soldier Mobility System (ISMS) input entities in a simulation, they read the

DIS network for the status of these other entities, and rely on their data for correct network interaction.[DRPRATT2]

2. Network Implementation

With ModSAF, NPSNET-IV.7 must coordinate the information for two logical networks to provide an accurate representation of events as they occur. To avoid duplication of icons, a filtering system is used to discard DIS PDUs from the DI_guy. All ISMS connections are socket-based point-to-point dynamically assigned based upon a central request port. This connection makes it necessary to have redundant communication for each displayed entity[DRPRATT2]. Each data unit for a particular movement once executed is sent out in the form below.

```
typedef struct {
        float    wrist[3],elbow[1],shoulder[3];//degrees of freedom
} ARM_ANGLES_TYPE;
typedef struct {
        float    toe[3], ankle[1],knee[3], hip[3];//degrees of freedom
} LEG_ANGLES_TYPE;
```

This is the type of information normally sent in a packet (not all inclusive), for each movement. Coordination of each move can become tricky if data locks and semaphores are not implemented before each modification and send to the network. To ensure consistent body orientation and posture, both ISMS and Jack inputs update the displays faster than the frame rate to account for the asynchronous nature of the SGI graphics pipeline and to achieve the minimal delay possible between action and display. ISMS sends out data as fast as possible (e.g., 30-60 Hz) to the DI_guy process, effectively overwriting any pending messages. The same is also done for the DI_guy back to the NPSNET-IV.7. This gives different cycle times between processes and reduces the apparent latency in the display. [DRPRATT2]

A specific discrete movement (posture change) such as going prone is accomplished by a button press, either on the keyboard or the throttle control while the user exercises a discrete transition. The simulated human executes a short script based upon the precomputed posture transition graph. As soon as the posture transition script starts, a packet is sent across the network so the correct behavior is displayed by all stations on the network.[SMPRATT]

General movement is updated each time through a main simulation loop. Each active icon (both local and remote) is notified to perform a moveDR operation. Upon receiving this request, the entity dead reckons its new posture based on its last posture, the amount of time since the last move/update and the dead reckoning algorithm and data defined by the remote entity. For the DI_guy, if a state PDU has not been received in two time periods (10 sec), the local object corresponding to the remote entity is deactivated and the entity is removed from the network.[PTBARH]

C. JACK ML IMPLEMENTATION

Jack ML was developed at the University of Pennsylvania to perform efficient high-resolution human motion [BADLER]. The Jack ML software used in NPSNET-IV.8.1 is a newer version designed by John P. Granieri et.al. for the Team Tactical Engagement Simulator (TTES) also presented at INCOMSS-94. This software package, the Jack Motion Library or Jack ML, has been integrated into NPSNET-IV.

The icon model used is actually a low-resolution model composed of 23 joints and 50 degrees of freedom. Hands are modeled as mittens, and only significant joints and joint combinations are present [SMPRATT]. The model is composed of 478 polygons, and has

no fingers, spine, eyeballs, or clavicle [JPGRAN1]. As seen in Figure 7, this model is still quite detailed.

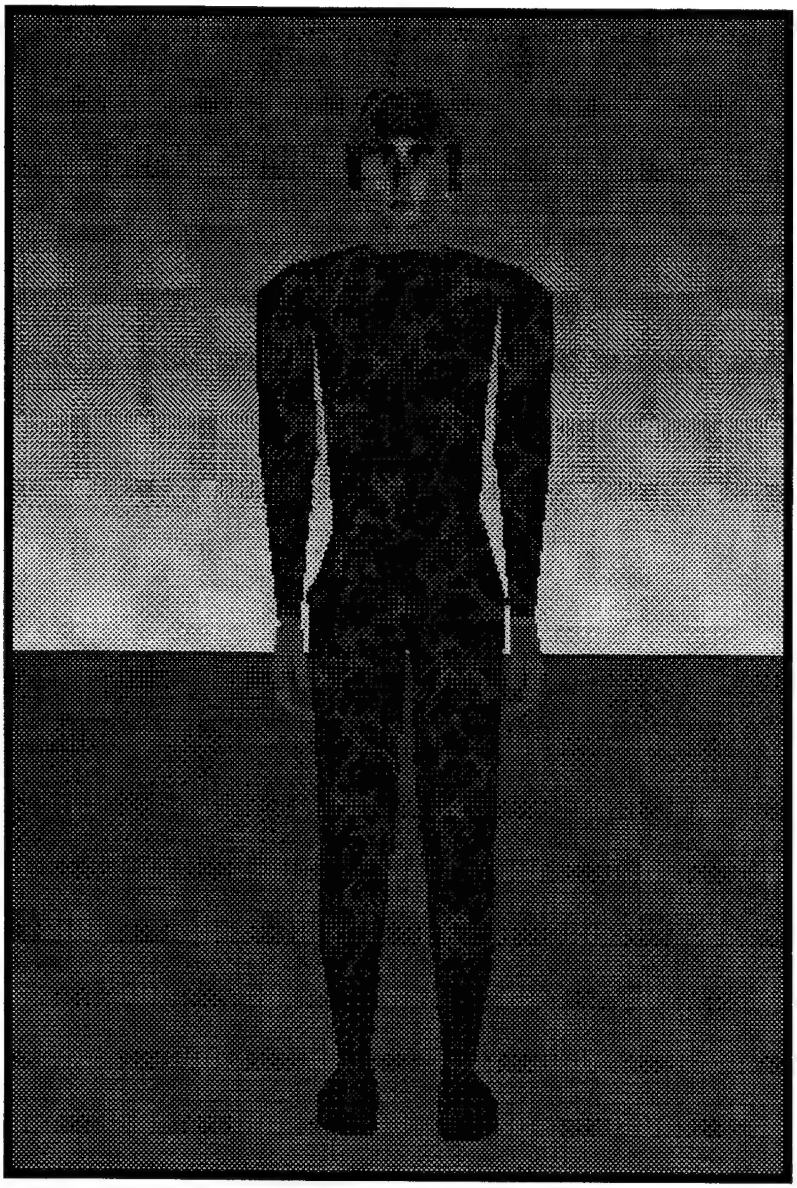


Figure 7. University of Penn.'s Jack ML Model. 478 Polygons

As Jack ML is used with a Head Mounted Display, HMD, and most icon activity is rendered at a significant distance from the view point, this scale of model is more than adequate for modeling human interaction in the foreground [SMPRATT]. The University

of Pennsylvania possesses more detailed Jack ML models used for fine grain, highly detailed human modeling which is beyond the scope of the requirements for NPSNET-IV.8.1.

Jack ML uses a set of scripted animations based on high-level behavioral and inverse kinematic constraints that are played back based upon the current posture of the simulated human, elapsed time, and motion pattern [JPGRAN2]. Jack ML changes postures, e.g., standing to kneeling, based upon scripted static posture states that translate the figure to a different posture. The Jack ML file is composed of 10-15 primitive motions to translate the icon from one posture to another. This makes implementation much simpler when transitioning from one posture to the next. As the original goal of the design was to provide a visual template for movement and not specifically to create the movement, scripting serves the purpose well.[JPGRAN1]

For dynamic posture changes, i.e., the velocity vector is not zero as in walking or crawling, a combination of scripted files is used to produce the effect. Walking, for instance, is begun when the velocity vector provided to Jack ML is non-zero. An initial step is executed to create a smooth transition from static to dynamic posture. Next, a cyclic state of scripted walking is executed and continues until a change in the velocity vector is noted, either a zero value or a change to run. When running, another set of cyclic state files to depict running are executed. A threshold value is set to return the icon to walking from running if this occurs.[JPGRAN1]

For NPSNET-IV.8.1, the animations used are modified in real time from user input. Since state transitions are scripts, and dynamic transitions are table-driven scripts, these can be overridden to more closely model a specific action. Currently, work is underway parallel to this thesis to add more mobility to the Jack ML model [SMPRATT]. Another scripted motion of a medic action is provided with Jack ML and is currently being implemented and further updated in NPSNET-IV.8.1.

1. Jack ML Control

Jack ML icons are controlled via several mechanisms. Speed and direction can be controlled by either keyboard or throttle and stick. Walking begins when the throttle is moved forward, advancing the velocity vector. By controlling the velocity and enumeration bits of the DIS PDU, Jack ML's motion is adequately modeled for dynamic postures. Similarly, static postures are executed via a button press on either the keyboard or throttle. As discussed before, this prompts the execution of a scripted transition to model user intentions[SMPRATT]. Real time modification of movement is used to move Jack ML's head which is tracked by the HMD. This adds to the realism that the user is in the virtual world as a human.

D. SUMMARY

In past virtual reality systems, human models have been represented using unrealistic static icons, or intricately designed icons which were computationally expensive. Recent developments such as the DI_guy, and Jack ML have greatly enhanced human interface in large-scale virtual reality systems. In order to further advance the human interface, more human icons are required per scene. The concept of level of detail modeling is not new, however, this is the first time this principle has been applied to and combined with human behavior to extend the capability of a system. LOD models are critical to increasing the number of human icons per scene while maintaining a high frame rate.

III. MODEL CONSTRUCTION

The premise driving this research was primarily speed, specifically frame rate. Jack ML was successfully implemented in NPSNET-IV.7J in December 1994. Jack ML is more realistic and anatomically correct than DI_guy, and it offers greater potential in simulating human movement. However when numerous instantiations of Jack ML are required, e.g., more than 20 entities, the computational cost of articulating joints becomes prohibitive. In order to overcome this limitation, level of detail models called Dude were developed. Current designs display the high resolution Jack ML model when distances between the user's view point and the human icon to be rendered are 100 meters or less. When the distance exceeds 100 meters, a Dude is rendered, instead of the Jack ML model, in one of its three level of detail models: Dude_medium, Dude_medium_far, or Dude_far.

A. IRIS PERFORMER API TOOLS USED

Dude models rely heavily on IRIS Performer API functions. Key to understanding the Dude model construction is a familiarity with the IRIS Performer functions pfNode, pfGroup, pfDCS, and pfSwitch.

1. PfNode

A pfNode is an abstract type. IRIS Performer does not provide any means to explicitly create a pfNode. Rather, the pfNode routines operate on the common aspects of other node types. These nodes include: pfGeode, pfGroup, pfDCS, pfScene, pfSwitch, and pfLOD; all of which are vital to the Dude models. Any IRIS Performer node is implicitly a pfNode, and a pointer to any of the above nodes may be used. Only a subset of the pfNode types actually contain geometry. These are known as "leaf nodes" in IRIS Performer.[IRIS]

2. PfGroup

A pfGroup is the internal node type of the IRIS Performer API hierarchy and is derived from the pfNode. A pfGroup has a list of children which are traversed when a group is traversed. Children may be any pfNode which includes both internal nodes (pfGroups)

and leaf nodes (pfNodes). Other nodes which are derived from pfGroup may use that pfGroup API. IRIS Performer nodes derived from pfGroup are: pfScene, pfSwitch, pfLOD, pfSequence, pfLayer, pfSCS, and pfDCS. PfAddChild appends a child to the group. PfRemoveChild removes a child from group. The bounding volume of a pfGroup encompasses all of its children.[IRIS]

3. PfDCS

A pfDCS (Dynamic Coordinate System) is a pfSCS (Static) whose matrix can be modified. A pfDCS can be used in place of a pfSCS, or pfNode. PfNewDCS creates and returns a pointer to a pfDCS. The initial transformation is the identity matrix. Movement, as is the case for the Dude models, can be set by specifying a matrix or translation, scale and rotation. Specific movements are then accomplished through the use of the pfDCS functions "pfDCSTrans" for translations, and "pfDCSRot" for rotations [IRIS]. These functions make Dude model movement possible.

4. PfSwitch

A pfSwitch is an interior node in the IRIS Performer node hierarchy that selects one, all, or none of its children. It is derived from pfGroup so it can use pfGroup API to manipulate its child list. PfSwitchVal sets the switch value of sw to val. Val may be an integer or symbolic token. It may take the form "PFSWITCH_ON" or "PFSWITCH_OFF" in which case all children or no children are selected.[IRIS] These calls are used to select the proper Dude model, as well as children of that model.

B. NODE HIERARCHY

Since interdependencies lie between the three models of Dude, it was possible to contain the information for all the models in one design structure. Best represented in Figure 8, a nodal hierarchy system was used. To summarize this structure, the highest node is a group node called "rootGP." It is this node that gets attached to the scene graph. Directly beneath this node is a DCS node called "hullDCS" which allows the distance-specific chosen model to be rendered to be moved to the correct position and orientation in the scene. Moving down below the hullDCS level finds a switch node. This first switch, called

"sw", is used to select between the Dude_far model and the other two. If the distance is sufficiently close enough to render a Dude_medium or a Dude_medium_far the switch is set to its first child. If the distance is sufficient to render a Dude_far, the switch is set to its second child. If the distance is too great for a Dude_far icon or if the model is not in the view frustum the switch is turned to off. The second child of "sw" is a DCS node called "far_bodyDCS" allowing Dude_far to be rendered and oriented properly in the scene. This node has only one child, a group node named "far_body" which contains the appropriate geometry to render Dude_far. The first child of the switch "sw", used only for Dude_medium and Dude_medium_far, is a group node called "torsoGP." Since both models have identical torsos the same geometry for both is used. The group node holding this common torso is labeled "body" which has no children of it own. Both models also share the same face and head. Since the head may be articulated, a DCS node is a child of the "torsoGP" node. This DCS node is called "headDCS" and allows the heads of two models to be articulated. HeadDCS has a child which is a group node containing the geometry of the head called "skull." The second child is also a group node containing the appropriate face geometry, either Rick or Duane, and is labeled "face."

The remaining four child nodes connected to "torsoGP" are for each limb. The structure for each limb is basically identical so only one limb shall be discussed. For example, the right arm is rendered for Dude_medium as follows: The right arm must be articulated so the first node connected to "torsoGP" is a DCS node called "armrightDCS." It has a child, a switch node called "sw2" which toggles between the Dude_medium or Dude_medium_far arm. If Dude_medium_far is selected the second child of "sw2" is rendered. This child is the arm of Dude_medium_far and its geometry is contained in a group node called "armrightmed." Since for this example, Dude_medium is selected the first child of "sw2" is used. This first child is another group node called "armrightGP" and has two children of its own. The first child is a DCS node corresponding to the right elbow joint called "armrightlowDCS." Attached to this DCS is a group node containing the lower right arm geometry for Dude_medium called "armrightlow". The second child of "armrightGP" is the right upper arm geometry stored in a group node named "armrightup",

the bicep of Dude_medium. This structure is repeated three more times one for the left arm and one each for the right and left leg.

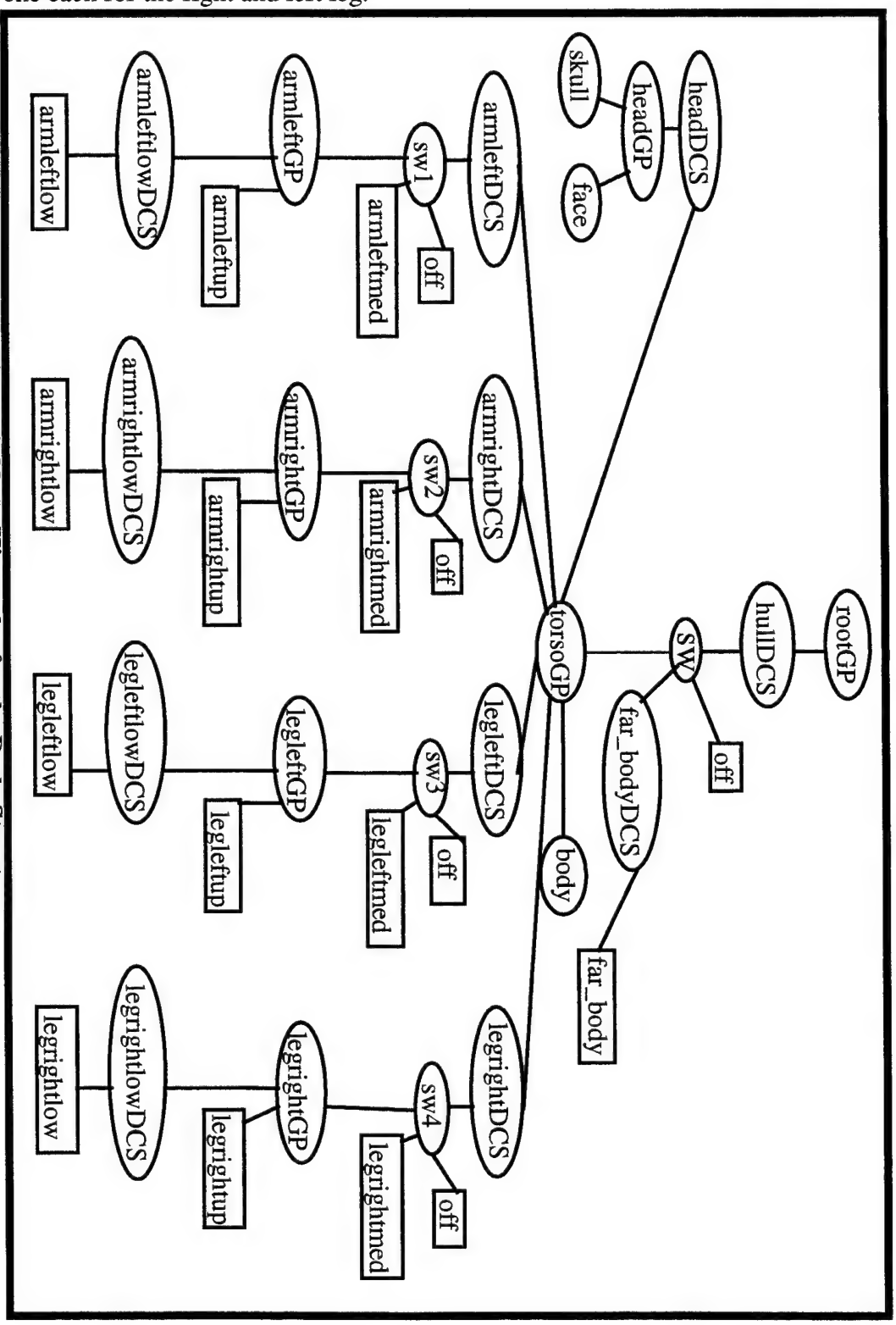


Figure 8. Node Hierarchy for the Dude Structure

C. MODELS

Three distance specific models were developed each representing a different level of detail. Using spatial-distance relationships as discussed in Chapter II, recall that as distance from the viewer is increased the ability to distinguish detail, i.e., individual joint movement and articulation, decreases. This perception continues to degrade with increasing distance such that the distance will reach a level such that no joints can be distinguished. This principle forms the foundation for the creation of the Dude models. All three of the models share the common property of uniform selection. They can be attired in either desert camouflage or woodland camouflage as the user dictates. Additionally, the two closest Dude models have faces which can be textured with one of two choices; "Rick" the friendly agent, or "Duane" the enemy agent, much the same as is done with the Jack ML model. Figure 9 shows all three models side by side. Similar in appearance, the total number of polygons per model drops significantly from the left model to the right model, with the left model, Dude_medium having 50 polygons, the middle, Dude_medium _far having 30 polygons, and the right model, Dude_far having three polygons. In this figure, Dude_medium has the face of Duane, and Dude_medium_far has the face of Rick. Tables 2 and 3 show polygon and articulation data for the various models in use.

Model Characteristics	High	Med-High	Medium	Low
Polygon Count	478	50	30	3
Total Joints	21	9	5	0
Total DoF's	50	19	12	0
LOD Range	0-100 m	100-150 m	150-200 m	200-350 m

Table 2. Polygon count and DOFs of Models

Articulated Joints	High	Med-High	Medium	Low
Head/Neck	2	1	1	0
Back/Waist	3	0	0	0
Shoulder/ Clavicle	4	2	2	0
Elbow	2	2	0	0
Wrist	2	0	0	0
Hip	2	2	2	0
Knee	2	2	0	0
Ankle	2	0	0	0
Toe	2	0	0	0

Table 3. Articulations per Model

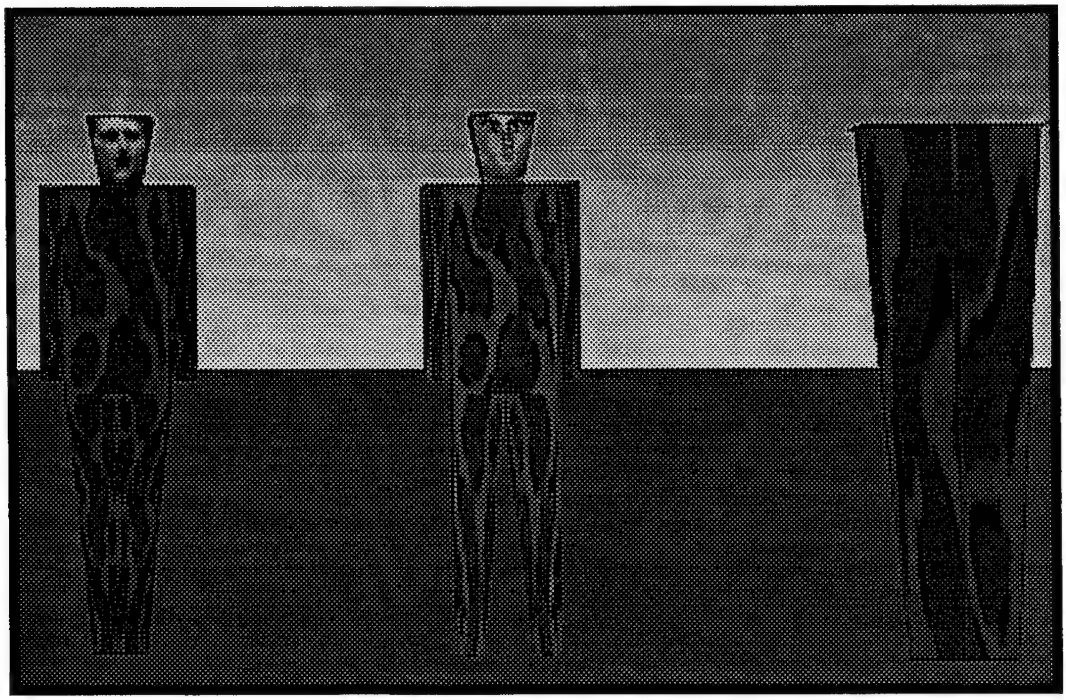


Figure 9. Dude Models

1. Dude_medium

Dude_medium is the highest level of detail Dude model. It is rendered when the viewer's eyepoint is from 100 meters to 150 meters away. Although it is clear that this model bears little resemblance to Jack ML in design, several basic Jack ML properties were integrated into the Dude_medium model. These include the physical dimensions of Jack ML to include height, width, and depth. Varying somewhat from the overall shape of Jack ML, Dude_medium has fewer joints and significantly fewer polygons; a primary goal of this design. Dude_medium has a total of 50 polygons reduced from Jack ML's 486. The joints or rotation points are as follows: Head, Left/Right Shoulder, Left/Right Elbow, Left/ Right Hip, and Left/Right Knee. The basic structure of Dude_medium is depicted in Figure 10.

2. Dude_medium_far

Dude_medium_far is the second highest level of detail Dude model, being displayed when the distance to the viewer is from 151 meters to 200 meters. Identical in design to Dude_medium, Dude_medium_far has the same physical dimensions as Dude_medium, and the only way to distinguish the two models is through the reduced number of joints. Dude_medium_far's rotation points are the Head, Left/Right Shoulder, and Left/Right Hip joints. Here again a significant savings from reduced polygon count was achieved through the elimination of secondary joints and additional polygons needed for articulation. For example, the arm is one piece versus two pieces for Dude_medium. This elimination of joints results in a model composed of merely 30 polygons and five rotation points as is depicted in Figure 11. Such a reduction is still perceptually effective and does not detract from the reality of NPSNET-IV.8.1.

3. Dude_far

Dude_far represents the lowest level of detail of the Dude models. Dude far is used when the distance from the viewer is 201 meters to 350 meters. This model is strikingly simple consisting only of 5 polygons and no joints. Dude_far has the same overall height and width as the two previous models, but its design represents a significant departure from

human characteristics. However, further extending the principles discussed in Chapter II regarding visual perceptions proves this design is still quite effective in conveying the image of a distant entity in the user's view frustum. Dude_far is composed of only three polygons, two along the x and y axis respectively. To be able to be viewed from above the third polygon was placed on top of the x and y polygons. With no joints, no articulations are possible nor required for this level of detail. As is depicted in Figure 12, the resulting model used to represent a distant human is successfully achieved with only three polygons.

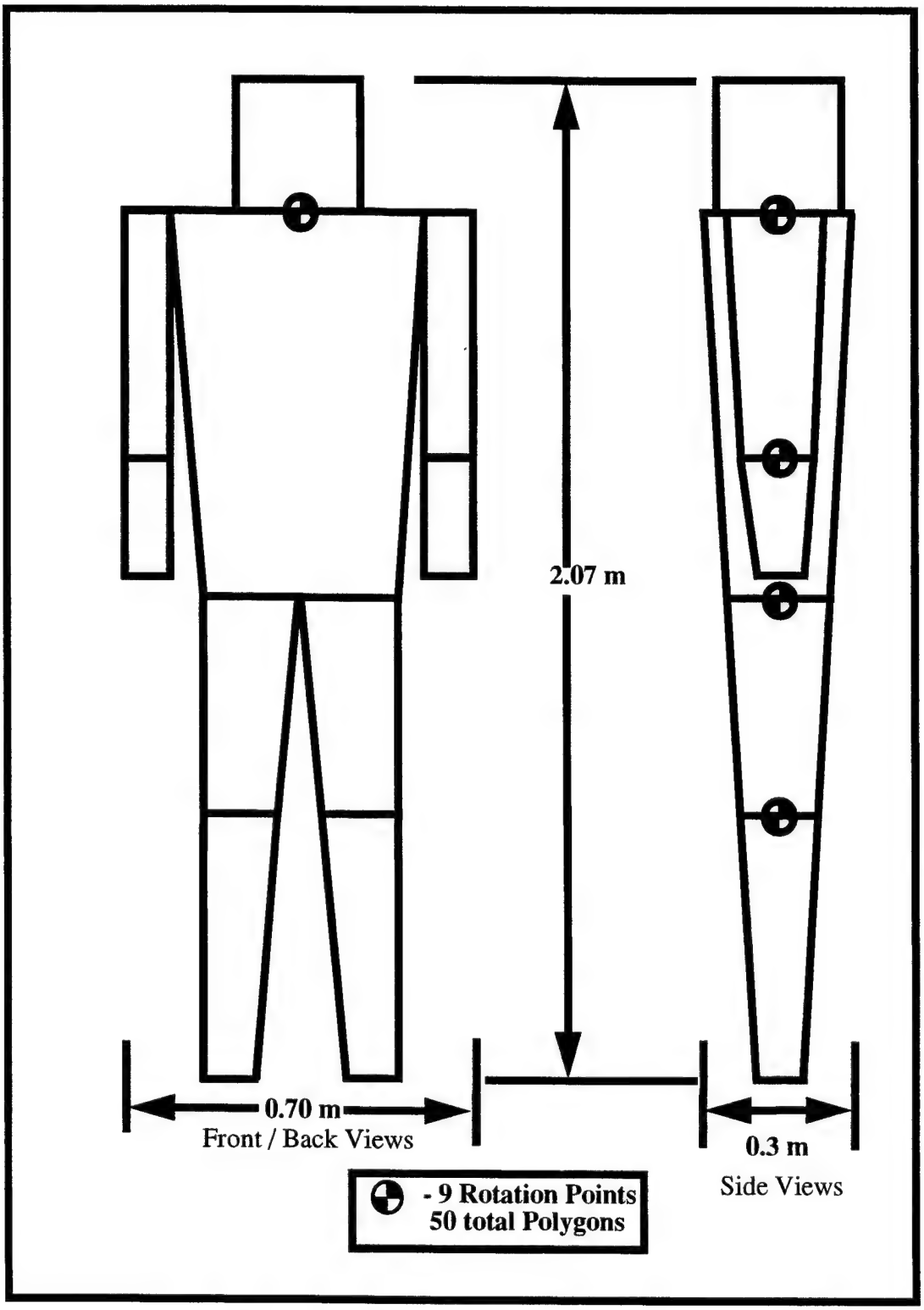


Figure 10. Schematic of Dude_medium

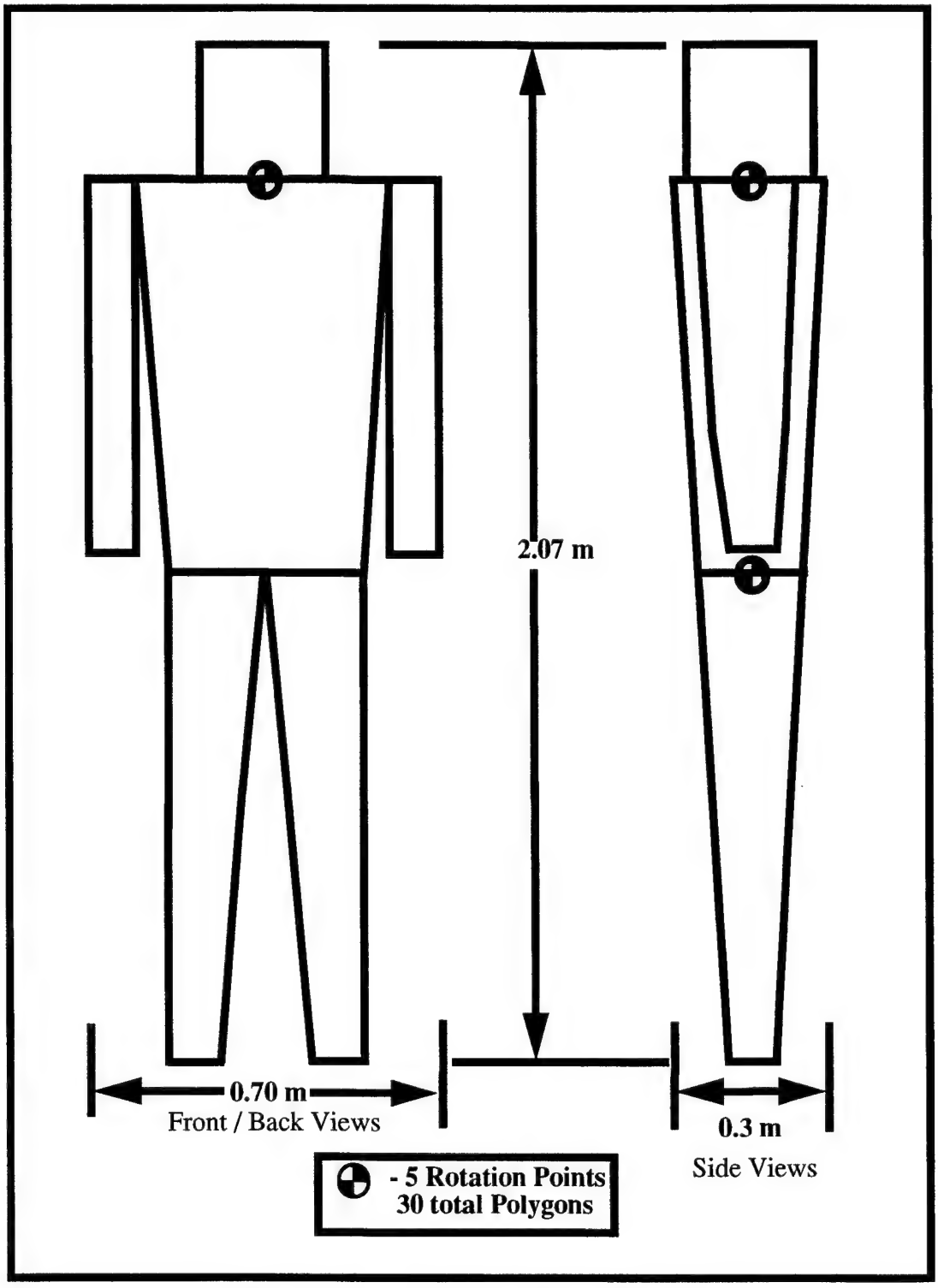


Figure 11. Schematic of Dude_medium_far

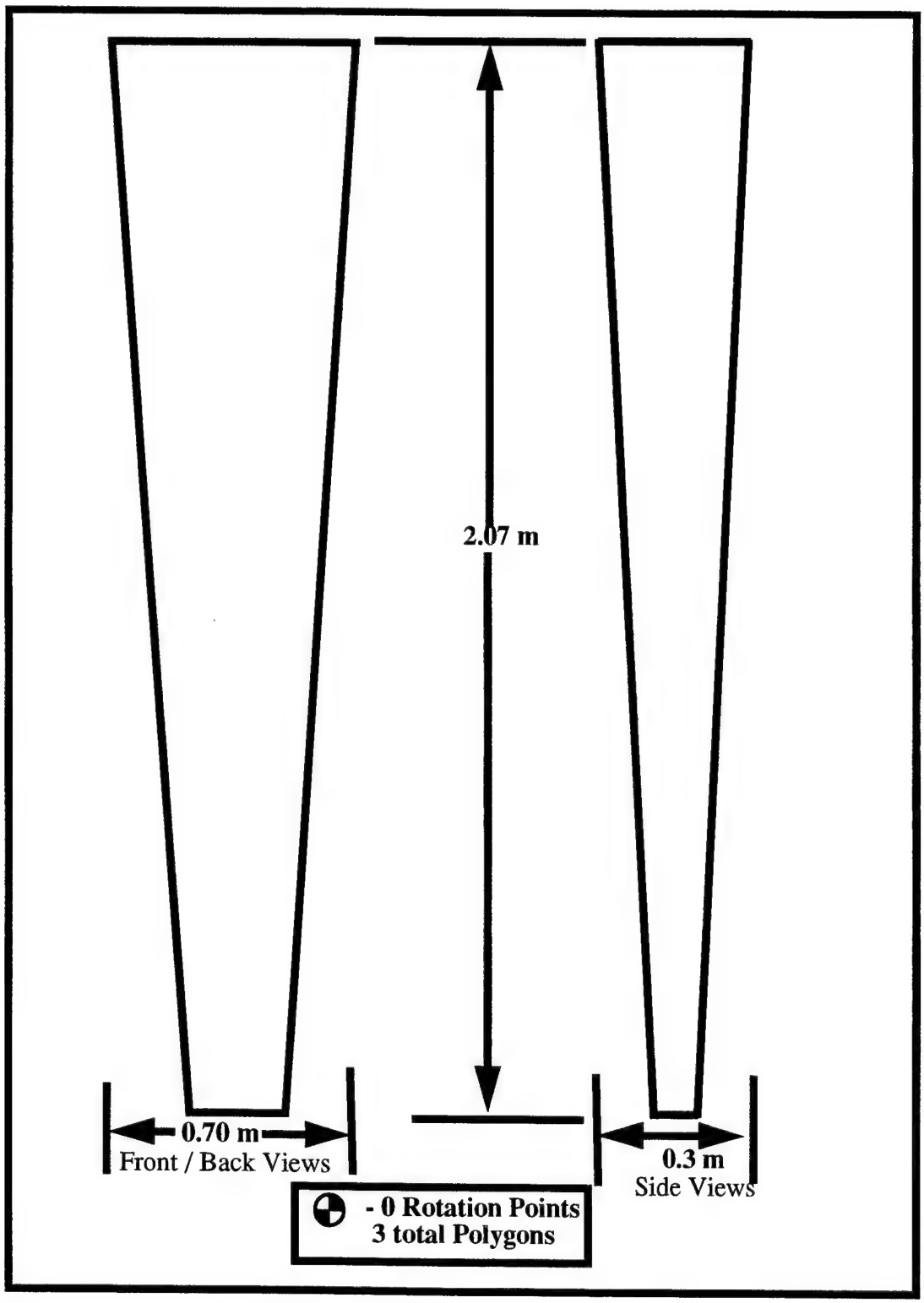


Figure 12. Schematic of Dude_far

D. CONSIDERATIONS

As discussed in Chapter II, IRIS Performer provides a built-in series of nodes and function calls to handle levels of detail in the form of pfLOD nodes. In the original node hierarchy design of the Dude models, we utilized these node types in place of the switch nodes now in use. Testing under various load conditions determined conclusively that pfSwitch nodes worked much better to preserve frame rates than did pfLOD nodes. Whereas the pfLOD node automatically shifts between models, the pfSwitch node required the creation of a unique algorithm to determine when the shifts should occur effectively replacing the pfLOD function. The algorithm used to perform this switching for Dude models is a simple extension of the distance formula. The two sets of x, y, and z coordinates used are the positions of the view frustum and of the Dude model. With this calculation the pfSwitch is set to the appropriate child.

1. The Dude Loader

To get the proper geometry into the group nodes of the IRIS Performer node hierarchy, a custom loader was developed consisting of files "dude_funcs.h" and "dude_funcs.cc". The loader is called with the type of Dude requested, either friend or foe. Space is allocated for the node hierarchy and then the loader uses three functions to build the three Dude models. The three functions each get the data to construct their models from three different data files, called within each function respectively. The models are inserted into the node hierarchy at the proper positions and as a result the entire Dude tree is constructed. The geometry of the tree is contained in a global structure defined in the file "dude.h." The structure is named DUDE_GEOM.

2. DUDE_GEOM Structure

DUDE_GEOM is the basic structure or node hierarchy to hold the geometry, i.e, vertices of polygons, texture coordinates, normal planes, etc., of the dude models. The node hierarchy explained earlier was placed in the DUDE_GEOM structure to save shared memory. In addition to containing the node hierarchy, DUDE_GEOM also contains all the

variables needed when the data files are read and manipulated by the custom loader portion of the program.

3. The Dude Class

The file "dude.h" also defines a class called DUDEClass. A class structure is required because we must be able to instantiate more than one Dude model at a time. It is also required because NPSNET-IV is C++ based. Without this structure, problems arise if more than one Dude is to be implemented and used independently; e.g., independent manipulation of a Dude model where more than one existed, was not possible. Using only one structure or node hierarchy meant that the information in that structure is common to all Dude models. This is sufficient if the Dudes are required to have the exact same positions and orientations. However, NPSNET-IV.8.1 requires multiple independently controlled Dudes. Thus a Dude class was necessary to achieve this control.

4. The Constructor

The action of the constructor is to clone the node system built by the loader for each instantiated Dude model. In this fashion, when a Dude is declared of type DUDEClass, i.e., an independent model, the entire node system is cloned and made available explicitly for that particular Dude. Also in the constructor the x, y, and z position of the Dude is established and the instantiated model is attached to the scene graph. Finally a single point is attached to the center of the Dude to be used for view volume culling, discussed in Chapter IV.

E. VARIABLES OF THE CLASS

The class DUDEClass was written in an object-oriented fashion and consequently maintains all of its own state information. This information includes all nodes of the type and number discussed previously that are required to properly be able to clone the structure DUDE_GEOM, its x, y, and z position with respect to the scene graph, whether it is inside or outside of the view frustum, the distance to the user's view frustum, and all the variables to control each of the models articulation points to perform its three movement functions of walk, kneel, and going prone.

1. Member Functions

DUDEClass includes its own functions to provide the Dude models with a variety of different movement capabilities. The class has a walk, kneel, stand, and prone function which work for all three different models irrespective of the model. Posture transitions are merely snap changes in which the model's posture is changed in only one frame. There exists no posture transition algorithm to display a Dude model as it shifts from standing to prone, for example, as is done with the Jack ML model. Posture changes for the Dude_far model are simple rotations and translations similar to the articulated models, but much simpler in design. The class also has the ability to attach and remove Dudes as children of the scene graph as well as the ability to place a Dude model at any x, y, and z position within the scene when required. As was discussed previously, the DUDEClass has its own function to determine if each Dude should be drawn or not and if drawn, which Dude level of detail model to draw. The following pseudocode shows the basic algorithm for selection and culling:

```
-pass channel
-test if PtIsecFrust
        ->yes   - calculate distance from frustum to model
                - switch to correct model
        ->no    - switch to off
```

2. Performance Figures

To test the Dude models a driver program was developed before the models were integrated into NPSNET-IV.8.1. This driver instantiated 200 Dudes and randomly placed them throughout a 500 meter by 500 meter grid. Level of detail switching between models was apparent in the test run. The goal of this testing was to achieve a maintainable frame rate of 20.0 frames per second. Average data for the numerous versions is summarized as shown in Table 4.

LOD Handling	Culling Method	Intersection Test	Frame Rate
pfLOD	pfAdd/RemoveChild	bounding spheres	0.5
		PointInFrustum	0.2
	pfSwitch	bounding spheres	15.0
		PointInFrustum	15.0
pfSwitch	pfAdd/RemoveChild	bounding spheres	1.2
		PointInFrustum	1.0
	pfSwitch	bounding spheres	19.0
		PointInFrustum	20.0

Table 4. Performance Data for 200 Dudes

IV. ENTITY CULLING

A. INTRODUCTION

While LOD usage is effective in maintaining a high frame rate, it may not be adequate by itself. With increasing numbers of entities in a system, e.g., tanks, aircraft, ships, missiles, etc., it becomes necessary to remove non-essential items or information from the active database so as to preserve frame rates in favor of the immediate view. Entity culling is useful for removing objects not visible to the user. In the case of the dismounted infantryman figure and its supporting LOD models, entity articulation and management become a computationally expensive task when multiple models are in the scene. By culling out entities that are not visible, we can effectively reduce the number of articulations required by removing and not managing the culled model. Important to the culling process is the point in the active environment that entities are culled. As culling can be an expensive calculation if not performed wisely, programmers must decide when it is economically feasible to cull an entity or continue to draw it. Removal of an entity in the Draw process is the most expensive operation, as the entity has already been sent through two pre-rendering processes. Simply managing the entity may be a less expensive option. It is possible to remove yet manage a model that may be just beyond the view frustum so that when it comes into view, the proper geometry is rendered. These two techniques are known as view volume culling, and range management. View volume culling is the simpler of the two, and it is most widely used.

B. VIEW VOLUME CULLING

View volume culling is achieved by testing the view frustum against a test object. As shown in Figure 13, an object is rendered if the test point or volume is within some range of the view frustum. If the object is in the frustum, it is rendered, otherwise it is not. As shown in Figure 13, icons A-E are present in the scene. Clearly, icon A is within the view frustum so it would be rendered. Icon B is to the right of the view frustum and would not

be rendered. Icon C lies beyond the defined far clipping plane of the scene, hence would not be rendered. Icons D and E present unique problems in view volume culling. Icon D is on the right boundary of the view frustum, so an algorithm and bounding volume must be used to determine if the object should be rendered. Since the bounding volume is partially within the view frustum but the icon is outside of the frustum, it represents a special case. Icon E presents a similar problem since it is in front of the defined near clipping plane. A similar approach to icon D is required for icon E. These principles are the foundation for view volume culling developed for the Dude models. In these models, we designed two test regions in order to discover which would produce better results. These test regions are a bounding sphere and a single point.

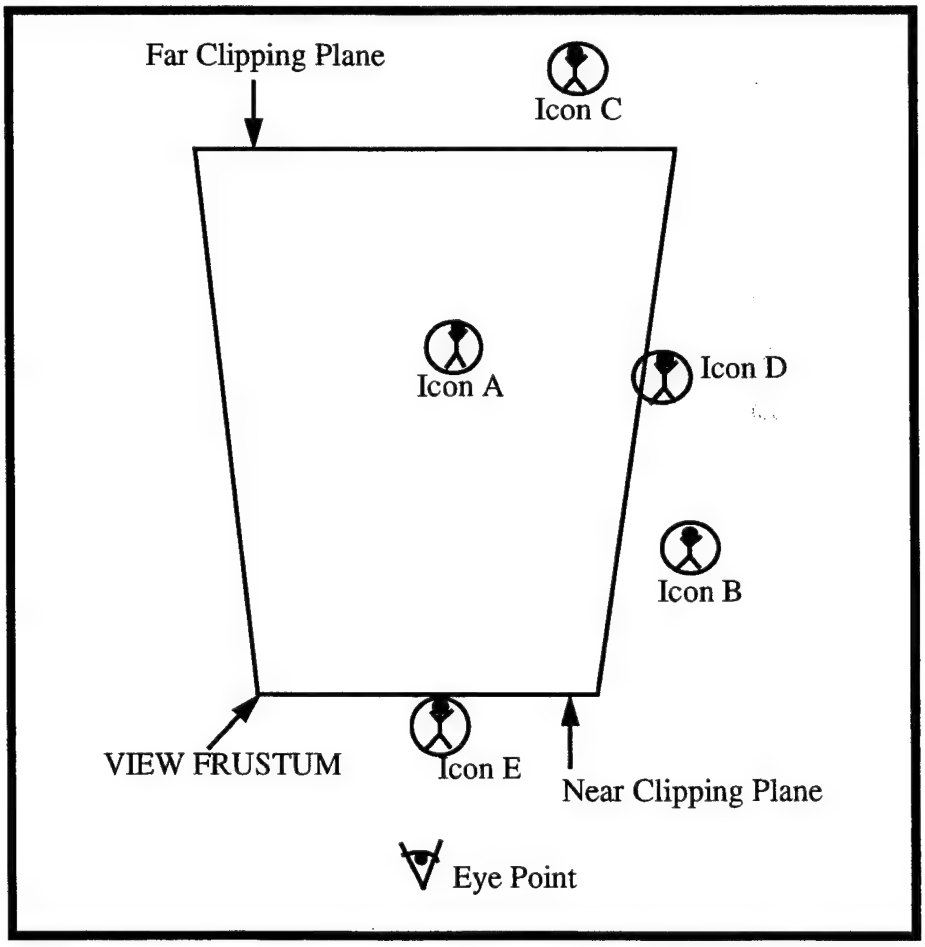


Figure 13: Basic View Volume Culling Testing

C. POINT IN FRUSTUM

Recall from Chapter III that for each instantiation of a Dude model, the constructor ensures that specific properties required for each model are included. One such property is the attachment of a single point to each Dude model for view volume culling purposes.The point attached to the Dude is of type pfVec3 so as to allow it to be attached to the exact center of the Dude, as shown in Figure 14. This point allows the usage of an IRIS Performer function call "pfPtInFrust" to determine if the Dude is in the view volume.

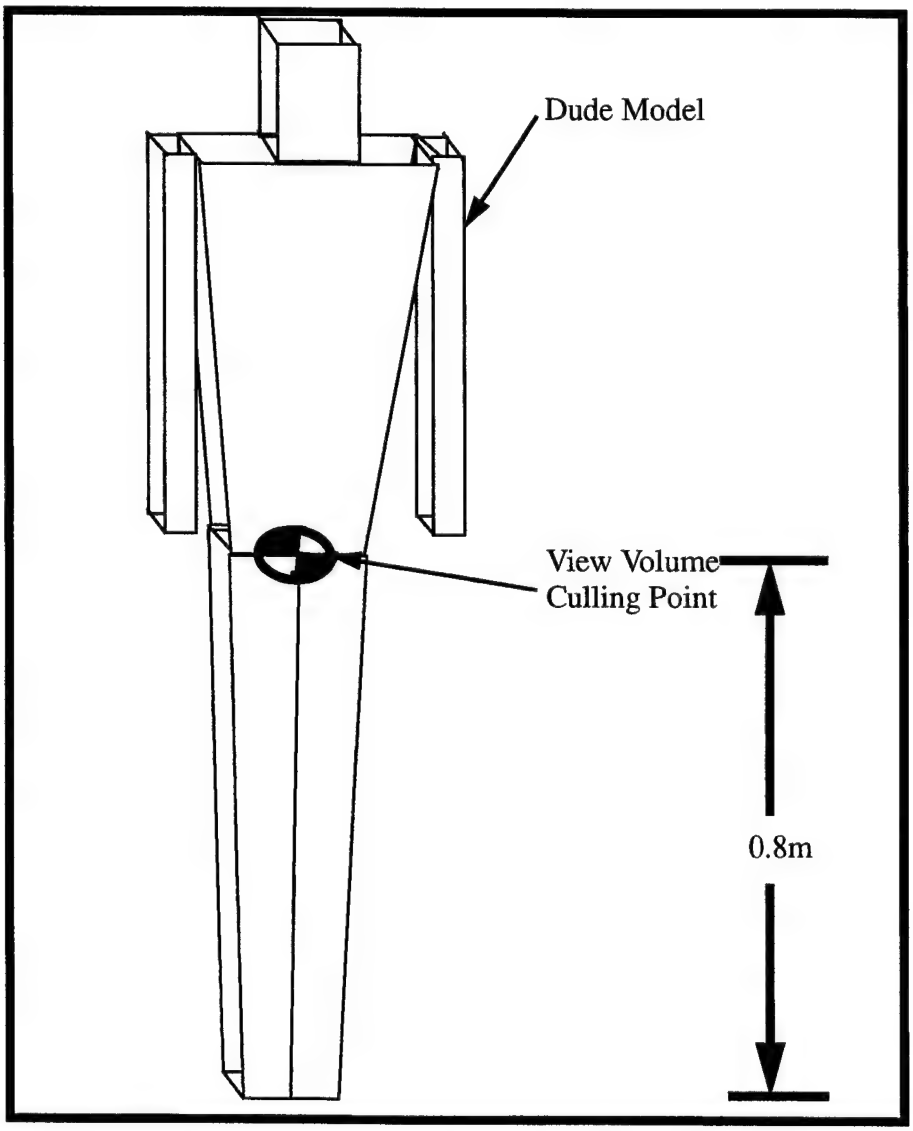


Figure 14: Culling Point of Dude using a single point

The "pfPtInFrust" function returns a Boolean and depending on that value the main pfSwitch at the top of the Dude node structure is set to either draw the proper Dude model or to draw nothing. Figure 15 shows culling using the "pfPtInFrust" function.

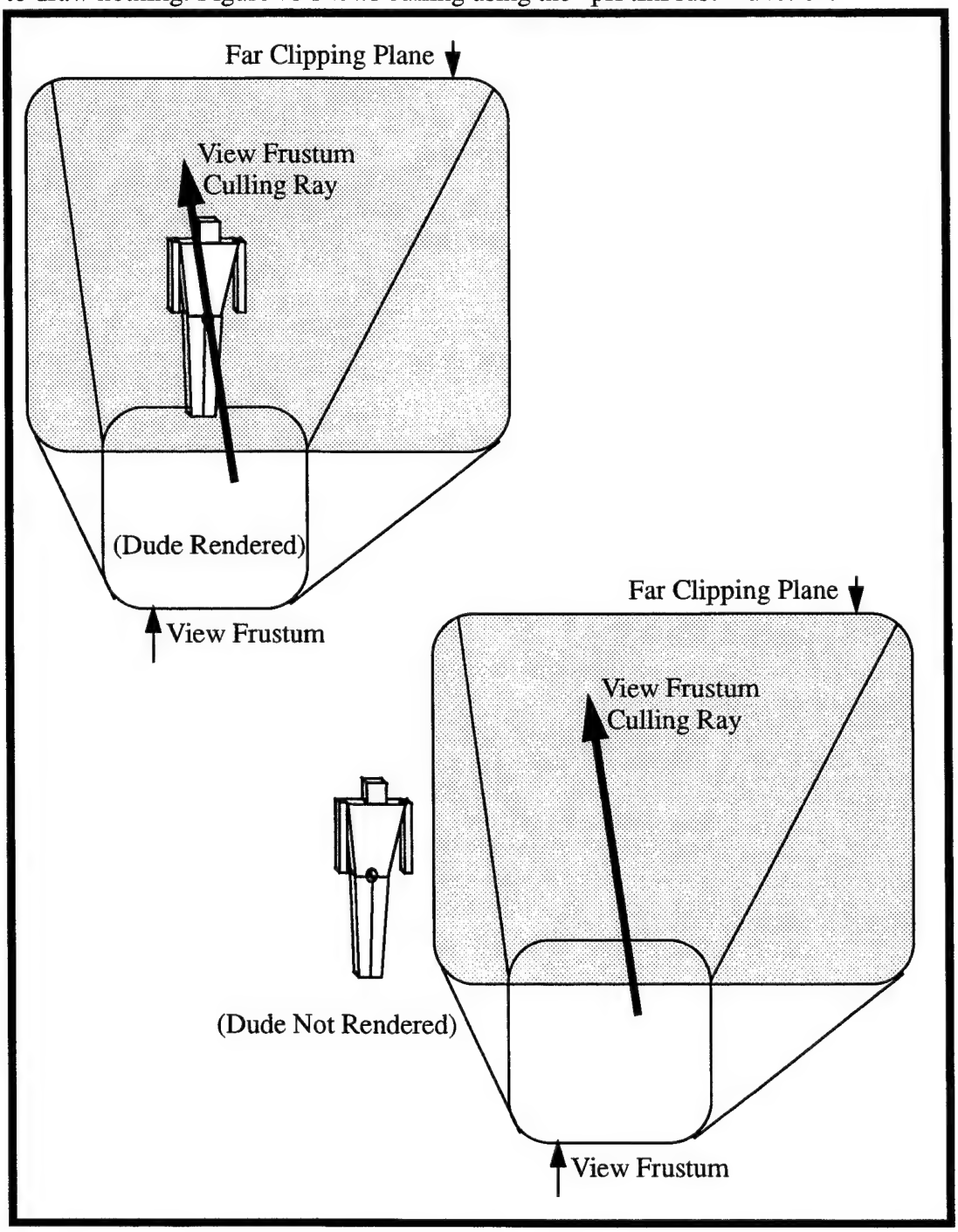


Figure 15: View Volume Culling using pfPtInFrust

The Boolean value returned by testing with "pfPtInFrust" is one of four values: "PFIS_FALSE" indicating the point is entirely outside of the frustum, "PFIS_MAYBE" indicating the point is partially inside or entirely inside the frustum, and "PFIS_TRUE" indicating the point is entirely inside the frustum. If the computation cannot be done trivially, the function returns "PFIS_MAYBE" as in the case of a point possibly inside the frustum. Since speed is paramount in culling, non-trivial solutions return "PFIS_MAYBE" instead of spending additional time determining whether the point is inside or outside of the frustum. [IRIS] This function result was easily used for our purposes to render a Dude model.

The "PtIsectFrust" method has its advantages and disadvantages. The advantage to the point method is that it is a single point having no distinct geometry. This makes its implementation simple in that no geometry must be passed for each instantiation. Its weakness is that because it is a point, it must be either inside or outside of the view frustum for culling to achieve its goal. This means that an icon may be partially within the view frustum and still not be rendered because the point is still outside. Conversely, an icon that should be culled out, e.g., one on the edge of the frustum, may be rendered until the culling point is outside the frustum. For both cases, a return value of "PFIS_MAYBE" is usually indicative of this condition, and is used to render the icon in our case. "PFIS_MAYBE" was rarely returned when using the "PtIsectFrust". Based upon our original driver program, not actually NPSNET-IV.8.1, and data shown in Table 5, the "PtIsectFrust" provided a slight computational advantage over the bounding sphere in the form of one frame per second increase during the initial loading. This increase was discovered while running the program with 200 Dude icons on the Onyx/4 R2 machine "Meatloaf".

LOD Handling	Culling Method	Intersection Test	Frame Rate
pfLOD	pfAdd/RemoveChild	bounding spheres	0.5
		PointInFrustum	0.2
	pfSwitch	bounding spheres	15.0
		PointInFrustum	15.0
pfSwitch	pfAdd/RemoveChild	bounding spheres	1.2
		PointInFrustum	1.0
	pfSwitch	bounding spheres	19.0
		PointInFrustum	20.0

Table 5: Performance Statistics for 200 Dudes

D. BOUNDING SPHERE

The second method examined in the course of this research was to utilize bounding spheres for visual culling. In a similar procedure to points, a bounding sphere was attached to each Dude model. Using another IRIS Performer function call "pfSphereIsectFrust" which also returns a Boolean, the pfSwitch is manipulated exactly the same as described above. Figure 17 shows the placement of the bounding sphere on the Dude model. Similarly, Figure 18 shows culling using the "pfSphereIsectFrust." Here again, the function returns a Boolean value having one of the four values discussed above. As the sphere is much larger than the point, "pfSphereIsectFrust" returns significantly more results of "PFIS_MAYBE," especially when a model is near or just beyond the edge of the view frustum. This result is easily tailored to render the icon.

The bounding sphere method also has its advantages and disadvantages. It has a distinct geometry that must be allocated and passed for each instantiation. If shared memory is critical, this could be more costly to the application. However, because this is a true bounding volume, it is much easier to predict its implementation when used for culling. If an entity is near a view frustum boundary, the sphere gives a more reliable return value than does the point-in frustum technique. The use of "pfSphereIsectFrust" was discarded after

testing which revealed that the point-in-frustum method offered a marginally better performance as shown in Table 5 above.

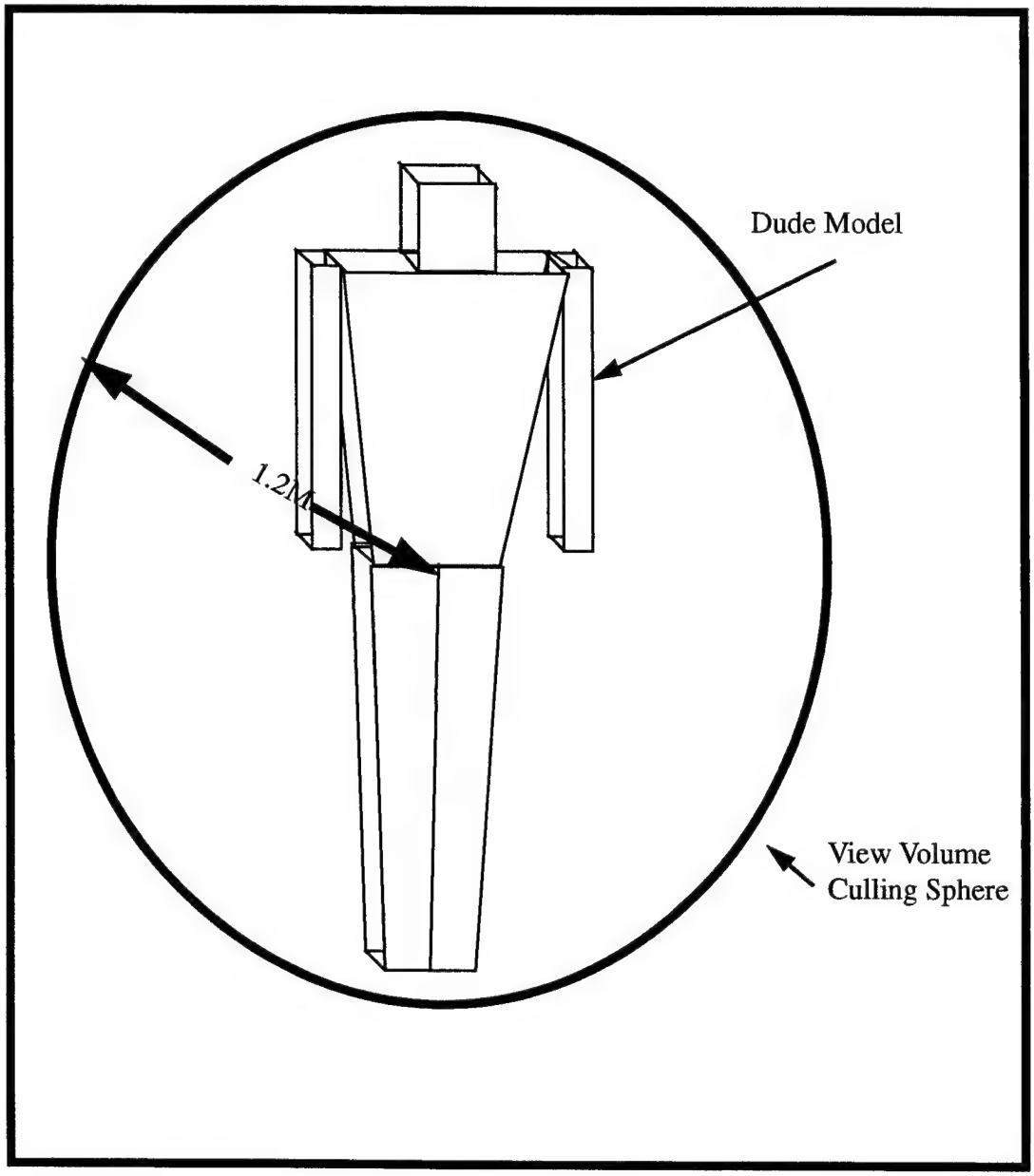


Figure 16: Culling Sphere for a Dude Model

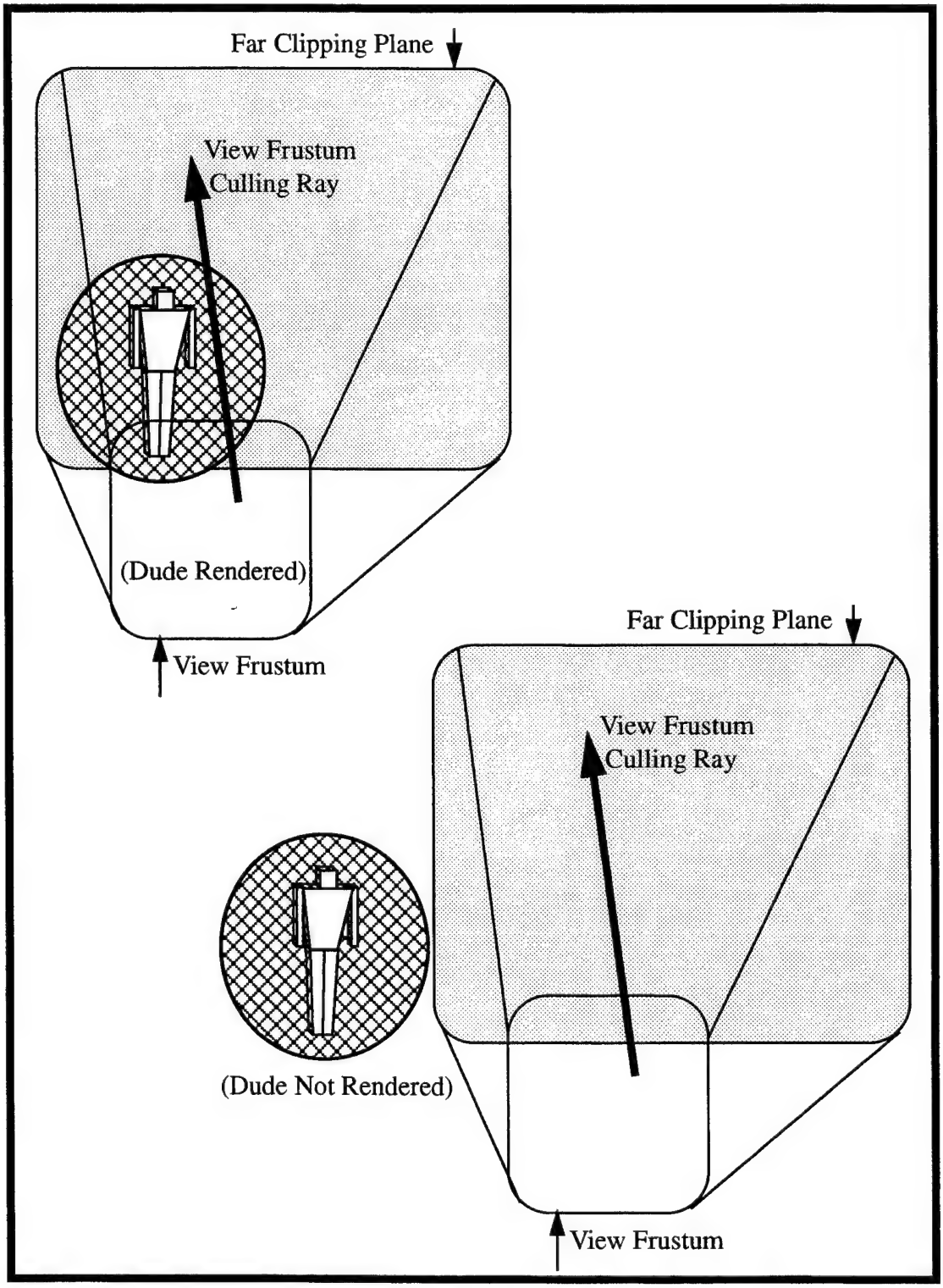


Figure 17: View Volume Culling using pfSphereIsectFrust

Figure 18 shows a human icon with only its shoulder on the periphery of the view frustum using both the point intersection test, and a bounding sphere. As this figure shows, the icon is in the same location in the virtual environment. It fails the culling test when using "PtIsectFrust", since the point is outside of the view frustum, and the icon is not rendered. It passes when using a bounding sphere, and the icon is rendered.

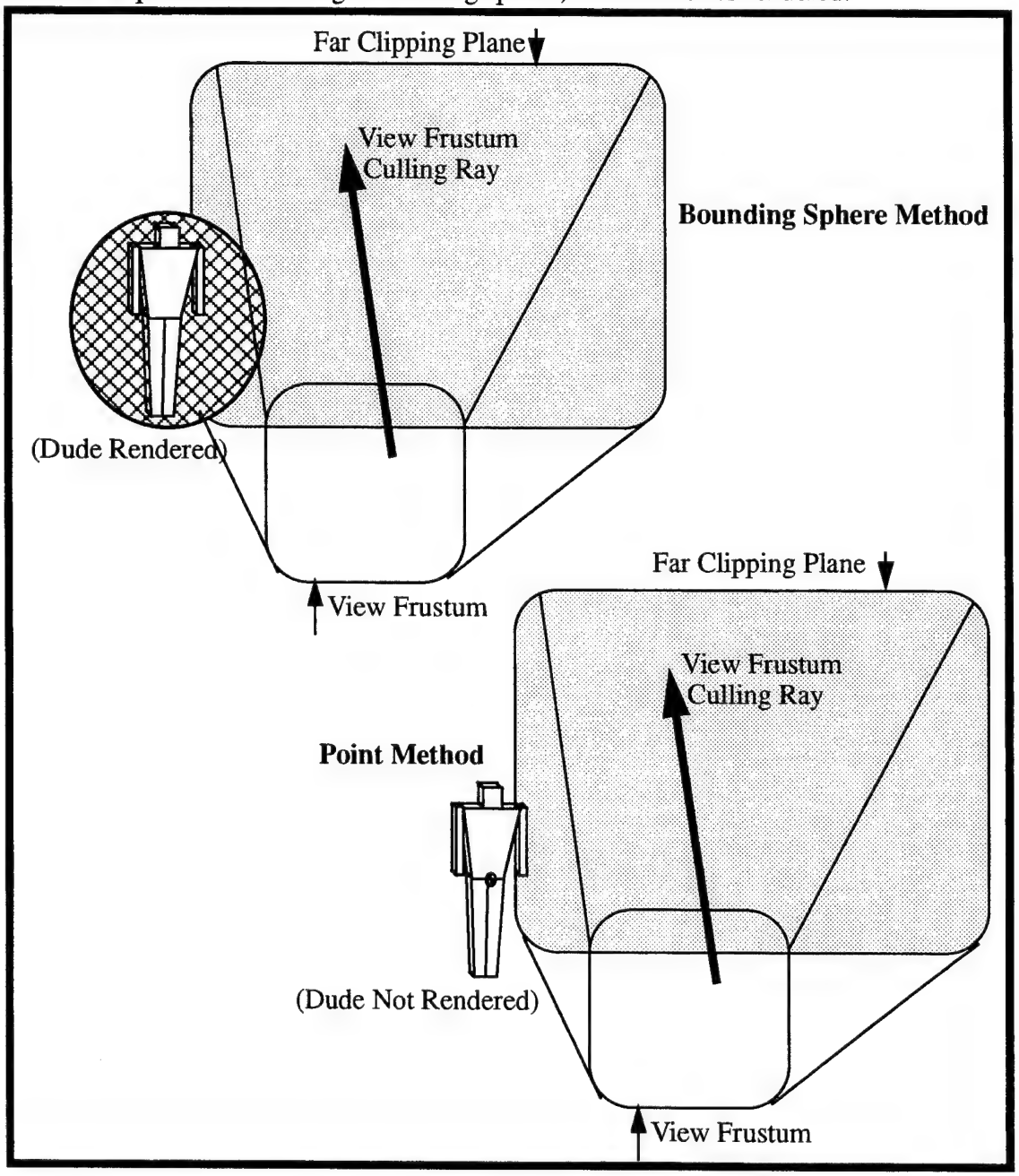


Figure 18: Point vs. Bounding Sphere in Culling

E. ADDING/REMOVING CHILDERN OF THE SCENE GRAPH

After determining that the point-in-frustum technique was better to manage the scene, an efficient method was needed to add and remove models from the scene graph. The initial development removed Dudes from the scene graph that were not in the view frustum. Models were removed using the IRIS Performer "pfRemoveChild" function call. Conversely when dudes needed to be added to the scene graph after returning to the viewer's frustum, the "pfAddChild" function call was used. Analysis of this method showed that, while useful, the calls were very expensive in terms of speed. Investigating other less costly methods resulted in utilizing "pfSwitchVal" function to set the pfSwitch node to "off" when a model was to be removed, and "on" when returned. When using "pfSwitchVal" the entity is still attached to the scene graph. When using "pfAdd/RemoveChild," the entity is removed from or added to the scene graph, expensive operations in terms of CPU usage. This expense is realized in reduced frame rates, the opposite effect we desired. After further testing, analysis of run data showed "pfSwitchVal" offered a marginally better performance as shown in Table 5.

F. SGI RENDERING PROCESSES

As shown in Figure 19, the NPSNET-IV.8.1 pipeline is composed of three main processes, the Application process, the Culling process, and the Drawing process. When culling entities, CPU and memory management gains are more efficient the earlier in the three-step process one can make them. For example, the most computationally expensive culling procedure is performed in the Draw process. Since the entity must pass through two rendering processes prior to reaching the Draw process, one must decide if it is economical to cull the entity or simple render it. The Draw process culling is a fine grain process, dealing with specific entities and what is in their view frustum. By the time an entity is being handled by the Draw process, memory allocation, pointers, and textures have been loaded and any manipulation of this icon will involve each attribute of the program. The Culling process, or simply Cull, offers a less expensive alternative to entity culling. If one can elim-

inate objects or icons at this level, much of the loading and allocation is saved before the process is passed on to Draw. Further up the pipeline is the Application, or App, where whole databases and textures can be eliminated to increase loading and rendering speed. For our purposes, NPSNET-IV.8.1 handles the App when loading textures and terrain databases upon entity instantiation. Icon range management, part of the Dude software, is performed in the App process. Finally, fine grain culling based upon the view frustum is handled in the Draw process.

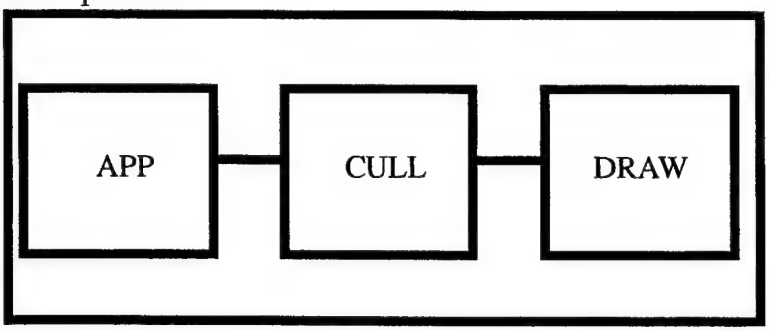


Figure 19: The Basic NPSNET-IV Pipeline

G. ICON RANGE MANAGEMENT

Range management was implemented in the Dude software in the interests of reducing the number of articulations processed for all entities present in a user's virtual environment. The concept of range management is to construct a multi-level monitoring system to determine when human icons should be managed, rendered, and articulated. This allows us to have multiple entities present in the virtual environment that are either managed and rendered, managed and not rendered, or not managed and not rendered. Figure 20 show the basic principles employed in the Dude software for range management. Here, as is discussed in section D above, range management takes advantage of the fact that it is less costly to remove entities in the App process than in the Draw process. For entities outside of the view frustum, they are not rendered thereby saving frame rates. For objects beyond the range of the frustum, in our case 260 meters, they are neither rendered nor managed so that no articulation calculations are computed further increasing frame rates. This represents a significant savings for DIS-based systems where every entity, regardless of its loca-

tion in the virtual world, broadcasts its location to all other systems. When entities come within the 260 meter range, they are then managed so as to be ready to render the proper geometry and activity should they come into the view frustum. Note that 260 meters was only used for testing purposes, and does not represent a fixed standard. This distance can be changed as a programmer desires for each specific application. Management of icons is achieved by setting up a range around the driven entity (the user), and determining based upon the velocity vector and heading each other entity in the environment, if they will be coming into the range of management or view. This calculation is performed for each DIS update cycle (normally 5 seconds) for each icon, whether it is the driven entity or one that is dead reckoned by the system. The difference between the range filter and the maximum viewing distance should be more than the distance travelled by the entity in one update cycle to ensure that the entities are managed before they enter the visual range [SMPRATT]. In the case of the far clipping plane, we have a 10 meter buffer in which a entity will be managed before it is rendered.

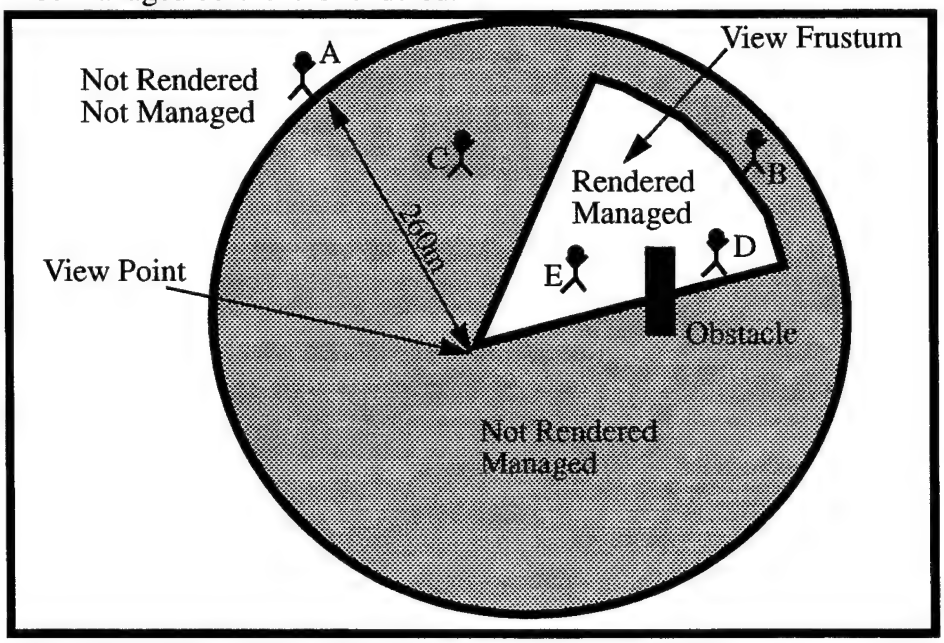


Figure 20: Range Management of Entities, after [SMPRATT]

From Figure 20, we can step through the multi-level custom culling algorithm and see the results. Icon A is outside the range filter distance of 260 meters so it fails to meet

the first level range check. All information concerning icon A is discarded. Icons B and C both lie within the range filter, but outside the view frustum. These two icons both fail the second level test, so they are only managed and are not rendered. Icon D presents a special case, since it is behind an obstacle. It is still managed and rendered, although it could be discarded using a line of sight calculation. In the interest of speed, we chose a liberal approach for this special case, and we manage and render the icon as long as it is in the view frustum. Of the five human icons, only icon E is clearly visible, and it will be rendered and articulated along with icon D.[SMPRATT]

H. SUMMARY

Entity culling is extremely effective and important to preserving frame rates. Although this is not a new concept, this is the first time it has been applied to articulated humans in NPSNET-IV. By removing non-visible entities, one can preserve frame rates by discarding objects outside of the view frustum. To further ensure frame rate preservation, range management is used to reduce the number of managed, articulated icons in the virtual environment. Both methods prove highly effective in maintaining real-time performance.

V. NPSNET-IV INTEGRATION

A. MODELS IMPLEMENTED

After extensive testing and development in our stand-alone driver program, the finalized Dude model designs and functions were integrated into NPSNET-IV.8.1. This integration was a major undertaking since all data for their design, LOD switching information, and all associated motion algorithm software had to be tailored to work in NPSNET-IV.8.1. The following three close-up figures show the final version of models implemented in NPSNET-IV.8.1. Figures 21 and 22 show the Dude icons in two snap transition postures available to the Dude models. Figure 21 shows the models in their kneeling postures, while Figure 22 shows them in their prone postures when viewed from above. These figures also show the models with the two uniforms, woodland green or desert tan camouflage, and caricatures, Rick or Duane, available. Figure 23 shows the three Dude models mid-stride when walking.

For each human figure in NPSNET, we use four different models to achieve multiple level of detail (LOD) polygonal representations. If the figure is not within the viewing frustum, IRIS Performer will prune the models from the scene graph during its cull process and no human figure will be drawn.(However, it will be articulated unless otherwise specified). When the figure is within the viewing frustum, Performer's level of detail selection and model switching functions will automatically trigger the appropriate model to be displayed depending on the pre-specified LOD distances and the viewing distance between the user's eye point and the figure. When the distance is small, the user is viewing a human figure which has full articulation of all the major body parts. However, when a user is viewing a human figure which is located off in the distance, the figure shown is actually much less complicated in appearance, and it has fewer articulated parts. Ultimately, as the viewing distance exceeds 200 meters, the human figure simply appears as a collection of just three polygons. During run-time, the user is unable to perceive a noticeable loss in figure

resolution because his ability to see details and articulated motions of objects decreases steadily as the objects move further away. To maximize performance, we chose to avoid the high costs of blending two different LOD models together at the LOD transition points and allow instantaneous model switches to take place. The results are still quite visually acceptable in a dynamic environment. Figure 24 shows all four human models used in NPSNET-IV.8.1 side by side, and views from various ranges.

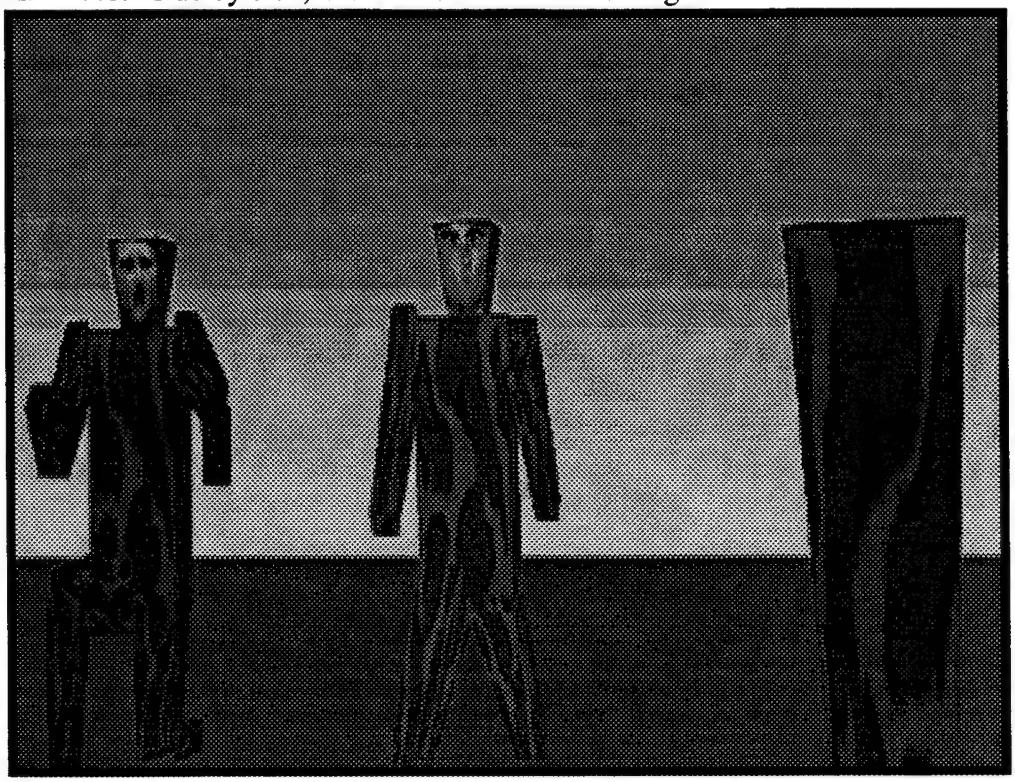


Figure 21: Dude Models: Kneeling Posture

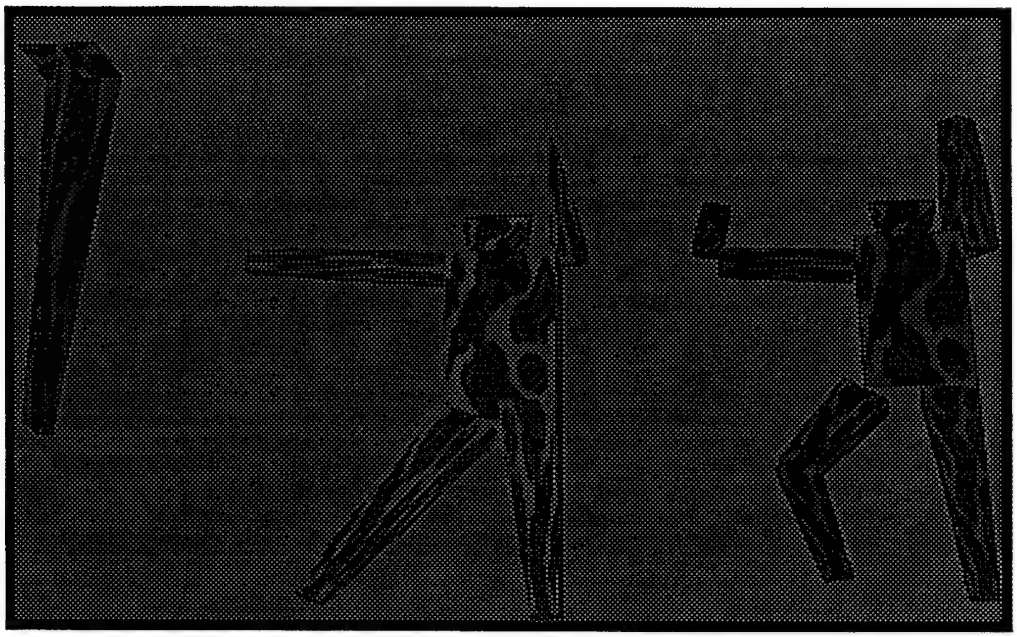


Figure 22: Dude Models: Prone Posture (as viewed from above)

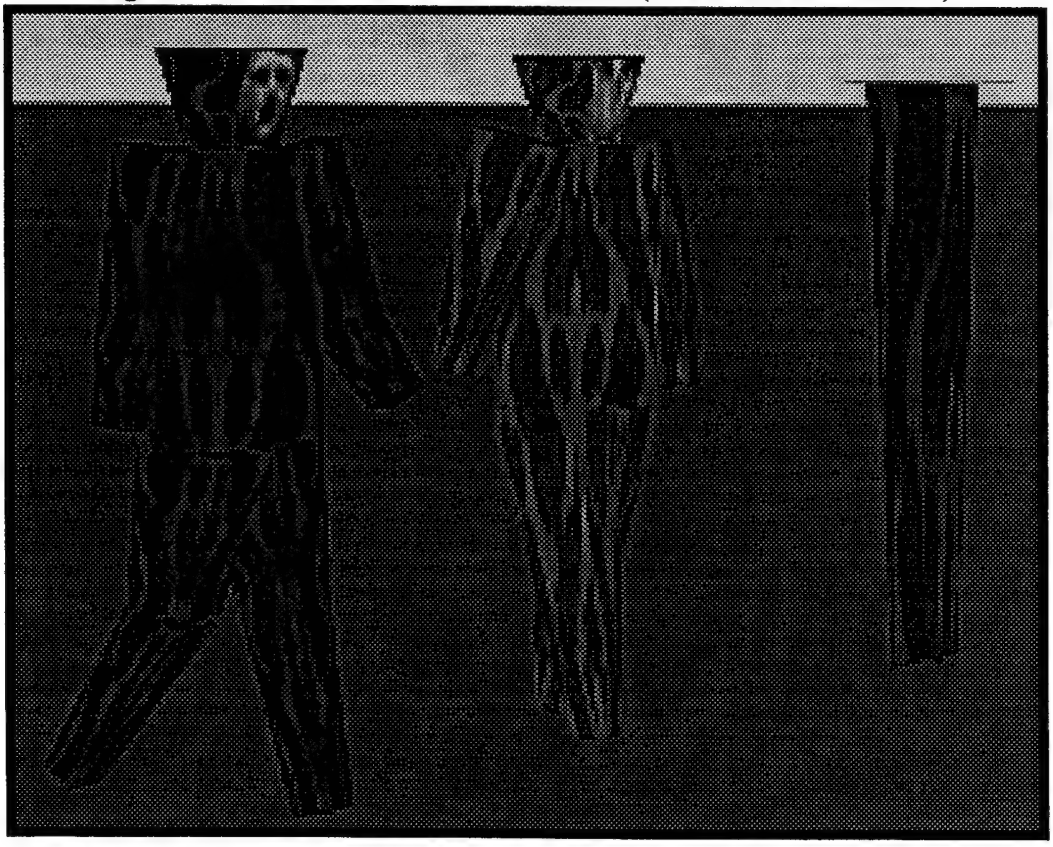


Figure 23: Dude Models: Walking

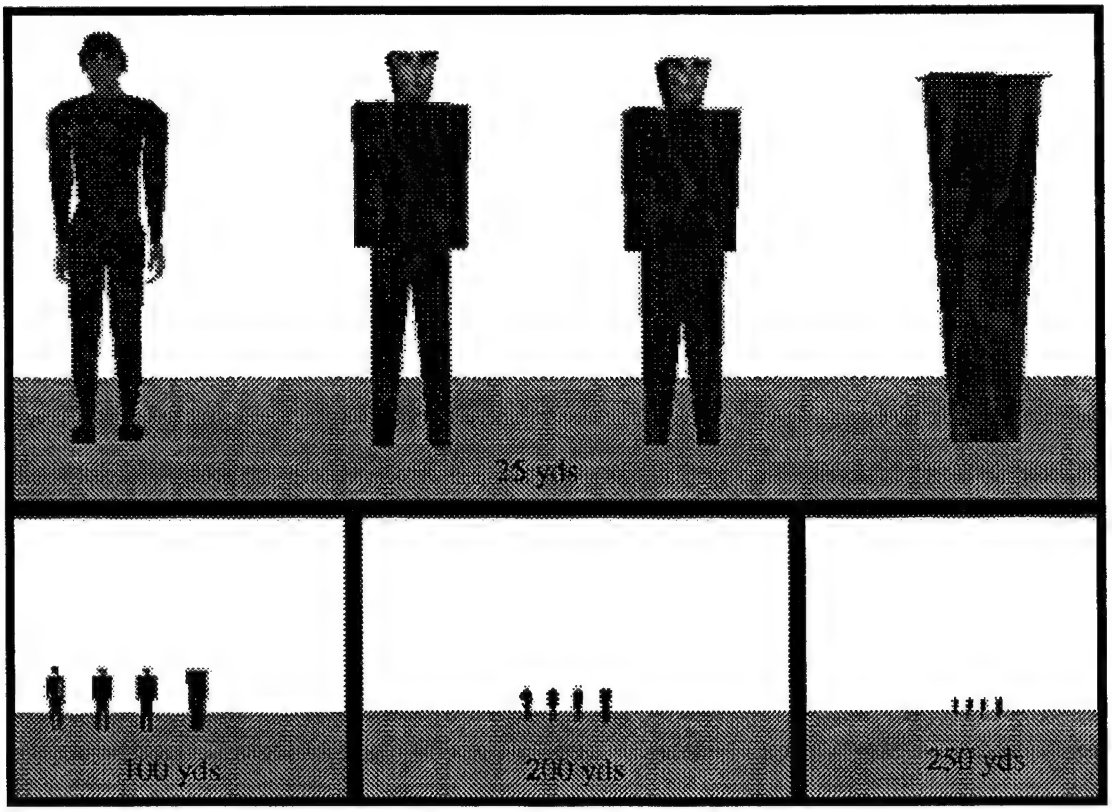


Figure 24: Comparison of Dude and Jack ML Models

The intention of the integration of the Dude models is not only to provide LOD support for the Jack ML icon, but also to provide a stand-alone Dude model in NPSNET-IV.8.1. Results obtained from our independent program, showed that a provision for a Dude-only mode of NPSNET-IV.8.1 would be useful in order to add more humans in NPSNET at the minimal cost to the system. For this reason, two distinct NPSNET entities with Dude support are available. These are: "dude," providing three LOD models composed of Dude models allowing Dude-only capabilities, and "jade" for full Jack ML support with Dude and Jack ML models.

B. SOFTWARE ADDITIONS/MODIFICATIONS

In order to properly integrate the Dude software into NPSNET-IV.8.1, several modifications to existing software were required. These modifications were made using the

SGI ClearTool, a new configuration management system controlling software revisions to NPSNET-IV.8.1. Using ClearTool, we had our own version of NPSNET-IV.8.1 to edit and test before fully integrating our changes into the final system. This new method worked relatively well, allowing us to make the necessary changes without affecting the existing system until we had completed the integration. As a new configuration management system, however, there were some "bugs" which had to be worked out in using ClearTool as well.

1. File Distribution

As the first step to integrating our software into NPSNETIV.8.1, we brought our entire thesis/help directory under configuration management by distributing our existing files into the appropriate directories. The header files (dude.h, dude_funcs.h, and jade.h) all went under "npsnet/src/entities/include" while the ".cc"(source) files (dude.cc, dude_funcs.cc, and jade.cc) went under "npsnet/src/entities/dude". The model data files (dude_friend.jcd, dude_enemy.jcd, dude_med_model.jcd, dude_medfar_model.jcd, dude_far_model.jcd) all went under "npsnet/datafiles/human".

2. Dynamic Data Addition

"Dynamic.dat" is a file containing all the dynamic models used within NPSNET-IV.8.1. This file was modified to include Dude and Jade models. In this file, a new unique letter code for both the Dude and Jade model is defined. Dude is assigned to "U", while Jade is assigned to "V". This allows NPSNET-IV.8.1 to read and load the associated data for various types of models. Currently NPSNET-IV.8.1 is able to load University of Pennsylvania "tips" files, and now our Dude and Jade models. The letter is used strictly by the model file loader routine in "netutil.cc" discussed below. For unique entity identification purposes, we also must assign a unique number to each model type. "Dude_friend" is 97, and "dude_enemy" is 98. Likewise, "jade_good(friendly)" is 99, and "jade_evil(enemy)" is 101. This model type identification number is made use of throughout NPSNET-IV.8.1 when a reference is made to a particular model or entity. Next, we assign the alive and dead

model number. The same number is used for all our models since we do not have a special dead model (we just manipulate the joints to make the alive model to look dead). After this, we assigned DIS-protocol specific identifications for the models. Assigned are a vehicle type, kind, and domain, a country code, category, designator, and model. Our friendly models are US while our enemy models are Soviets. Each of the models are assigned weapons by adding munitions lines under the model type definitions.

Recall from Chapter II that entities in NPSNET-IV.8.1 are a part of a large class hierarchy. As shown in Figure 25, driven entities are children of the general class type "Vehicle". Of interest to this research is the vehicle subclass "Person". Dude, Jade, and Jack ML are subclasses of the "Person" class, from which they each derive their general velocity and physical attributes. Below the subclass "Person" level, each model has defined its own specific attributes.

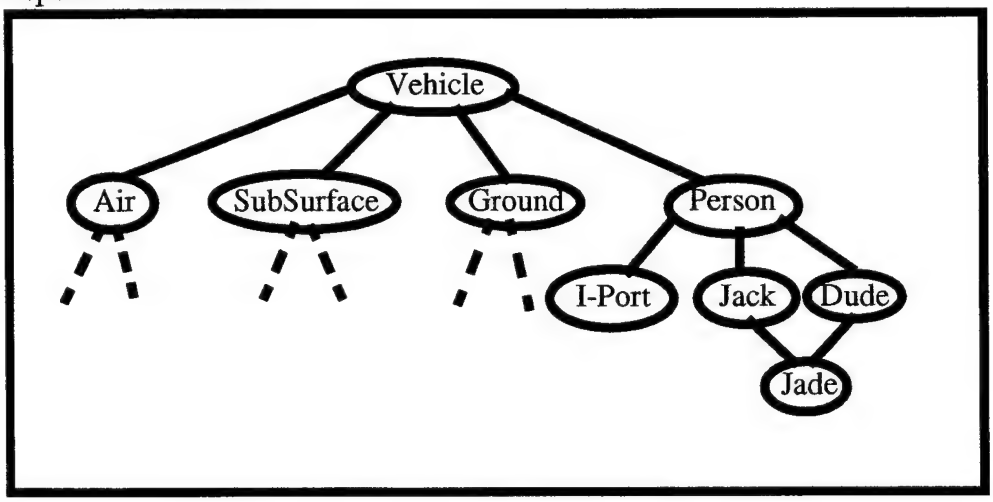


Figure 25: NPSNET-IV.8.1 Entity Class Hierarchy

Finally, for all entities in use in NPSNET-IV.8.1, there are enumerations defined for each vehicle class type. The enumerations uniquely identify all of the class types of various entities in NPSNET-IV.8.1. With the addition of Dude and Jade, two more enumerations were necessary. Dude is assigned as 12 and Jade is assigned as 13.

3. Loading Models into NPSNET-IV.8.1

The "Setup-Vtype" table routine in "netutil.cc" is responsible for loading all the dynamic models from "dynamic.dat". Statements were added to handle letter cases "U" and "V" for the new human models. When starting NPSNET-IV.8.1, all entity models are loaded into the system database. Next, the user specifies which model to select, i.e., dude_enemy, dude_friend, jade_evil, jade_good, which prompts NPSNET-IV.8.1 to clone the geometry and structure of the selected model. At this input, the corresponding identifying tag for the model is either "U" or "V" which prompts "netutil.cc" to read the "dynamic.dat" model lines discussed above. For the case "U", the cloned model data is passed to a table of vehicle types which keeps track of all the information of that particular model. This indicates that the alive model of this vehicle is our node structure. Also inherent to this structure is the creation of a rectangular bounding box for basic entity collision detection. A bounding box is created for each vehicle entity in NPSNET-IV by mathematically determining the overall size range of that entity. Finally, supporting textures for the Dude models we supplied to the NPSNET-IV.8.1 list of textures.

For the case of "V", there is a slightly different procedure. As shown in figure 26, we create a pfSwitch called "which_human" and attach the Jack ML model as the left child and the Dude model as the right child. Since both models were previously loaded they are added to the pfSwitch with the pfAddChild function. Above the switch we create a pfGroup

called "jade". This is required because NPSNET-IV.8.1 clones each model as it is instantiated and the clone process only clones a pfGroup. Finally we attach this switch to "jade".

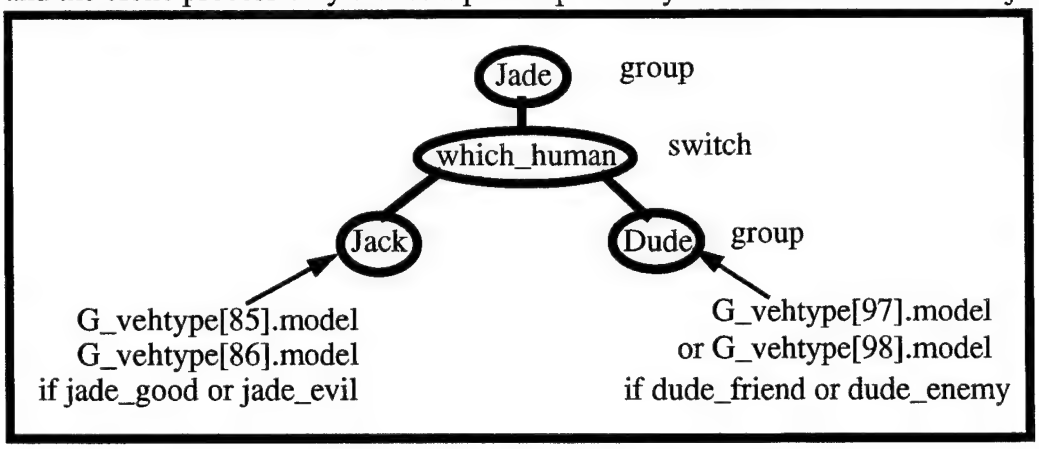
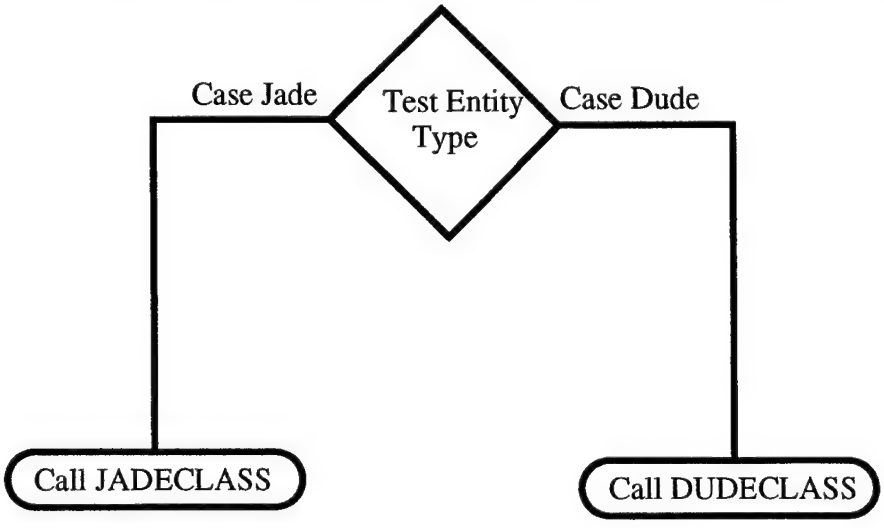


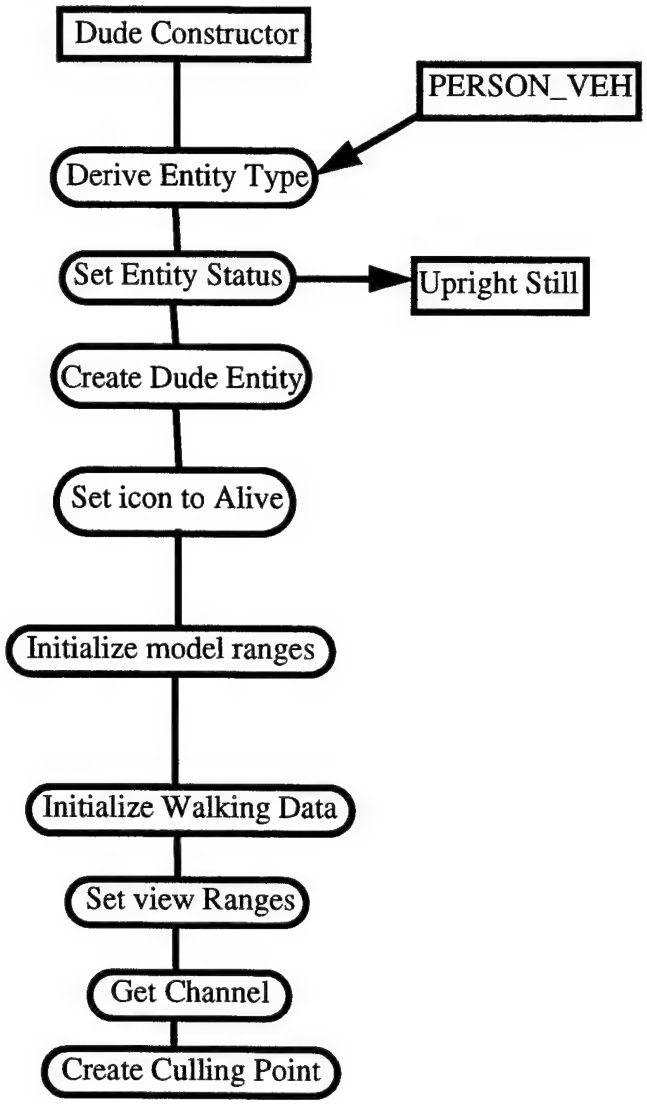
Figure 26: Switch Structure of Jade

4. Creating an Instance

In "entity.cc", we added the capability to create an instance of our new entity types. Runtime selection clones a copy of the appropriate Dude model and attaches the top most node of DUDE_GEOM, rootGP, to a dynamic coordinate system node which is added to a group of other moving objects in NPSNET-IV. This function was added to the existing function "create_new_entity" for the cases Dude and Jade. Under each case, that particular class is instantiated. The following flow chart shows the basic structure of the new function.



Each of these calls its respective class DUDECLASS or JADECLASS constructor function. "Dude.cc" provides the constructor functions for creating and managing the Dude models. These models function in a stand alone mode in NPSNET-IV.8.1. A flow chart for their creation is as follows:



As shown above, Dude models are derived from the NPSNET-IV PERSON_VEH. The initial Dude model posture is set to standing which is handled by a flag setting it to upright and still. We then call the function "create_entity" which clones the DUDE_GEOM we created via our loader in "netutil.cc". This cloned structure is the Dude model provided

to the user. When a Dude model is created a cloned copy is made. Each clone is independent and has its own set of variables, joints, functions, etc. This cloned Dude model is attached to NPSNET-IV.8.1 by the use of a pfDCS called "hull" as shown in Figure 27. The user's Dude model structure is the grandchild of hull.

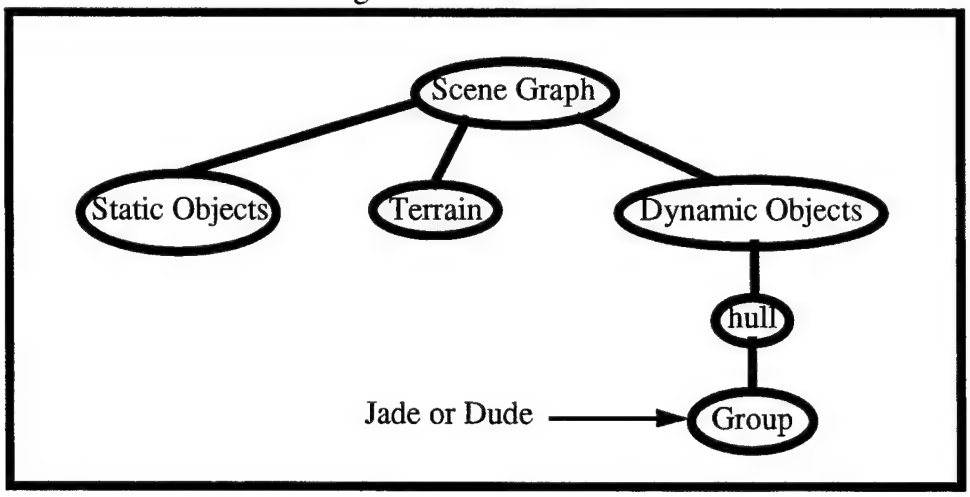


Figure 27: Attachment of Dude Structure to Scene Graph

NPSNET-IV.8.1 uses hull to move the Dude model to the correct coordinates in the virtual environment and to keep track of where the Dude model is located. Hull has a child of pfSwitch which has as its left child the Dude alive model, and as its right child an empty node. Because DUDECLASS has its own "create_entity" function which is called automatically, we find all of the DCS and Switch nodes of the cloned structure for each Dude model instantiation. In "dude.h" we created our own DCS and Switch nodes for each instantiation and we provide this to the respective node in the cloned structure by attaching the DCS nodes to the cloned structure and passing the main node the new model structure. These functions set the main_DCS (declared in dude.h) to the "hullDCS" as was shown in figure 8 (Chapter III). At this point, we set initializers to display and articulate the Dude model. This is done in the definition "displaying_model = TRUE" and "dead = FALSE". Then we set an inherited variable, "num_joints" to 9, since Dude_medium has 9 articulation points. Next, all the walking variables are initialized. Following this, we set the ranges for the three Dude models to 50, 100, and 250 yards. Note that these are 100 yards less than the distances

used with the Jade models supporting the Jack ML model. After this, we get the view channel of NPSNET-IV.8.1 so as to perform entity culling.

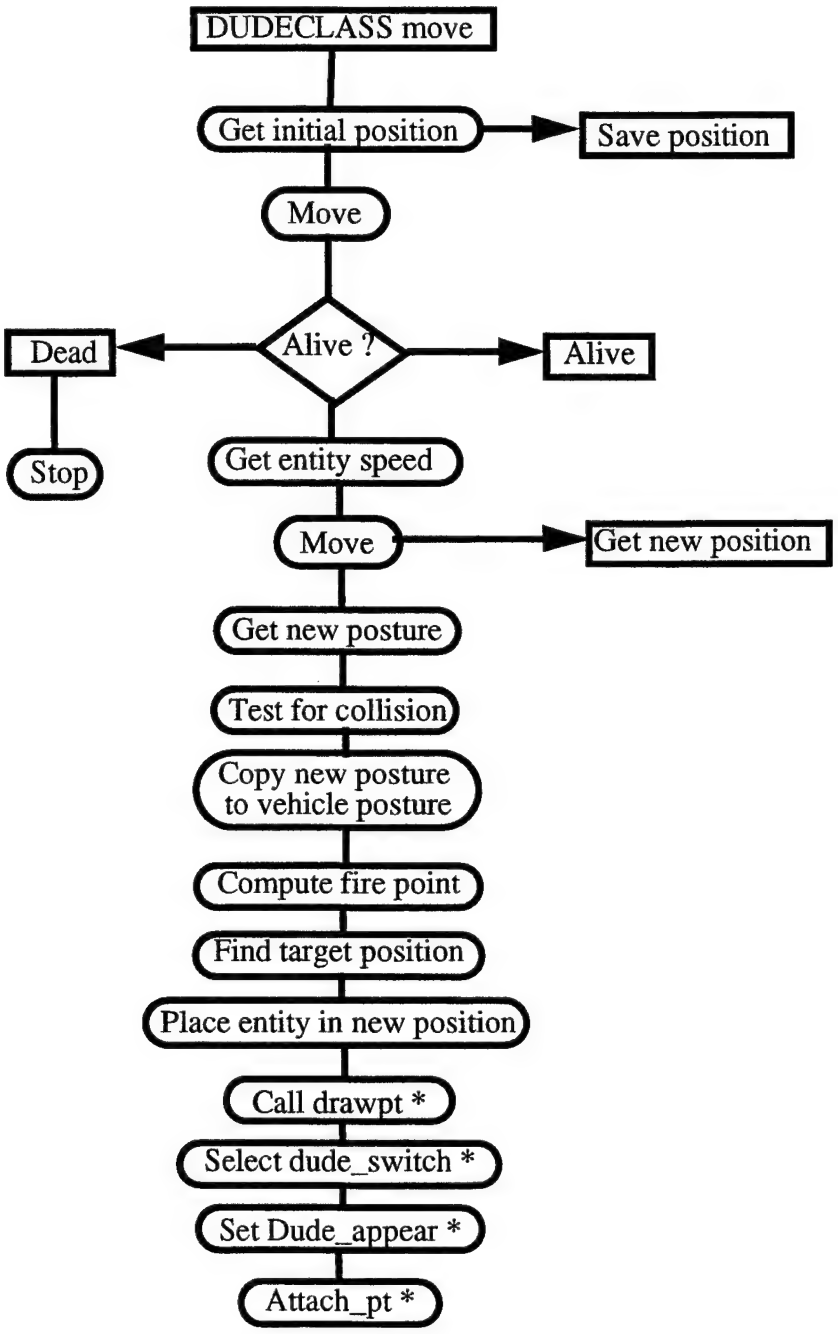
5. Manipulating the Object

For each frame, NPSNET-IV.8.1 updates all of its entities by calling their respective update functions: "move" for the local driven entity, and "moveDR" for both the driven and the remote entities. "Move" provides the locomotion and collision detection functions for the driven entity.

a. "move"

"Move" takes its input from the user based upon what type of entity the user represents. At each NPSNET-IV.8.1update cycle, "move" provides PDU information to the network informing all remote sites of the user status. Each time through the main simulation loop, each active entity (both local and remote) is notified to perform a "moveDR" operation. Upon receiving this request, the entity dead reckons its new posture based upon its last posture, the amount of time since the last request, the last move/update and the dead

reckoning algorithm and data defined by the remote entity. The following flow chart shows the main points required for the driven entity manipulation:

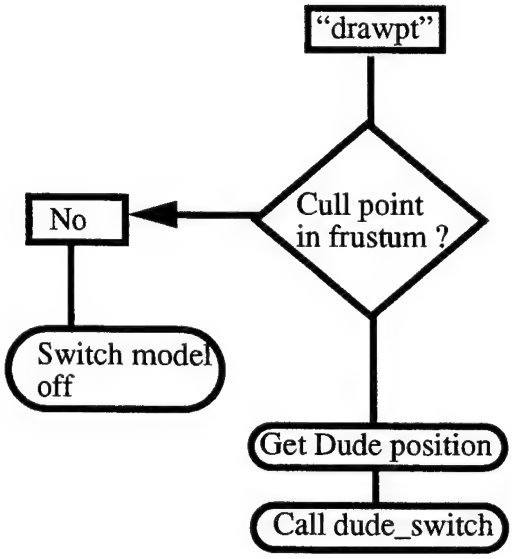


In general, the "move" function first finds the entity's initial position. Next, "move" calculates the new position of the driven Dude entity based upon a time interval, velocity, and heading. After moving, the function tests for collisions. Following the moving

functions, we next perform the custom culling functions. Taking the rest of "move" at each function marked with an "*" we step through the actions that occur next.

b. "drawpt"

At "drawpt", we check to see if the Dude is in the viewing frustum. If the Dude model is not in the view frustum, we switch it "off" as shown by the following flow chart:

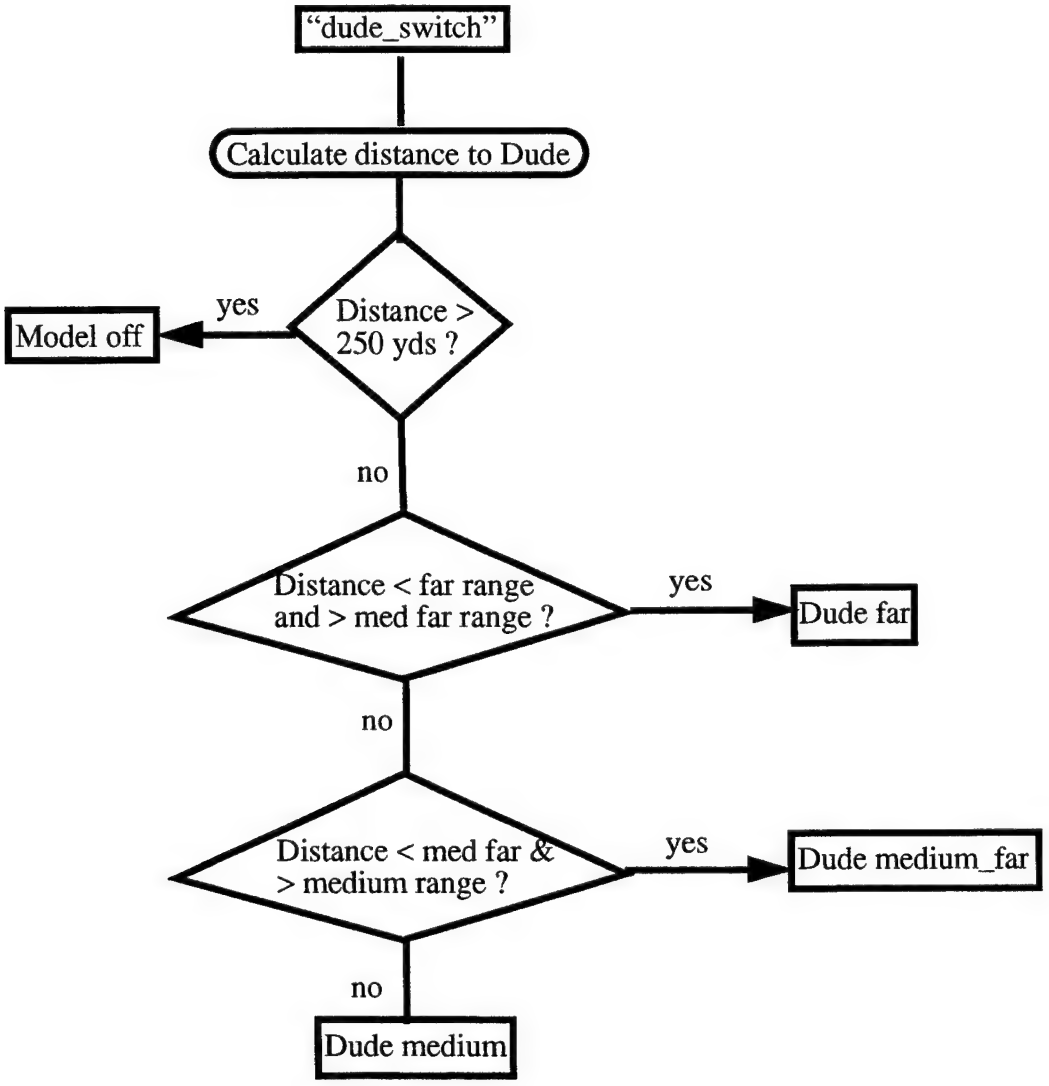


If the Dude model is being displayed and the "PtIsectFrust" test discussed in Chapter IV shows the culling point is now outside of the view frustum, the Dude model is culled out by the setting the pfSwitch to "PFSWITCH_OFF". Likewise, if the model is not being displayed but now should be, the model is returned to the display by setting the pfSwitch to the current model in use.

c. "dude_switch"

At the function "dude_switch", we find the distance from the viewer's eyepoint to the Dude icon using the Euclidean distance formula and toggle the pfSwitches to display the right Dude LOD model. The following flow chart, called in the "move" func-

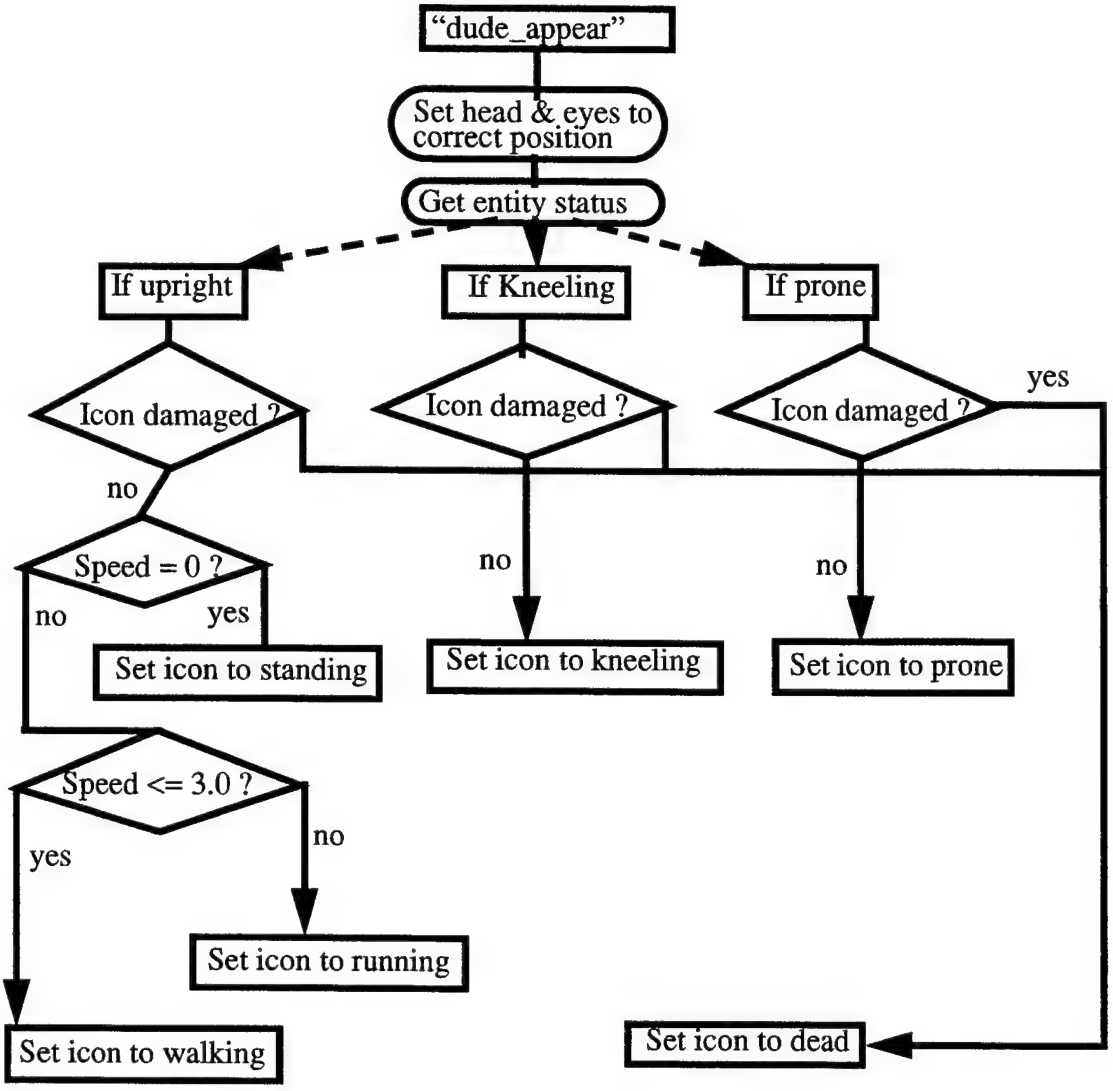
tion, shows the location of the distance calculations and LOD switching functions we use to change Dude models.



In the calculation above, we test the distance from the eye point to the Dude icon to see if the icon has moved beyond the model rendering distance of 250 yards. If so, the icon is switched off. Next, we test the distance for the proper LOD model to display.

d. "dude_appear"

At the function "dude_appear", we determine the user's desired posture to display. The following flow chart shows the algorithms used:



From the above flow chart, it is possible to follow through the various maneuvers to call posture changing functions for the Dude models. At each call, the icon is manipulated using "pfDCSRot" and "pfDCSTrans" to achieve the desired posture. Note that in the above flow chart, speed ranges are defined for the Dude models. A velocity of

0.0 indicates the icon is not moving. For velocities between 0.1 and 3.0, the icon is walking. Velocities between 3.1 and 10.0 are provided for rendering running Dude icons.

e. "attach_pt"

"Attach_pt" is a short function that places a culling point for a Dude icon in the same location as the moving entity. The following pseudocode shows the construction of this function:

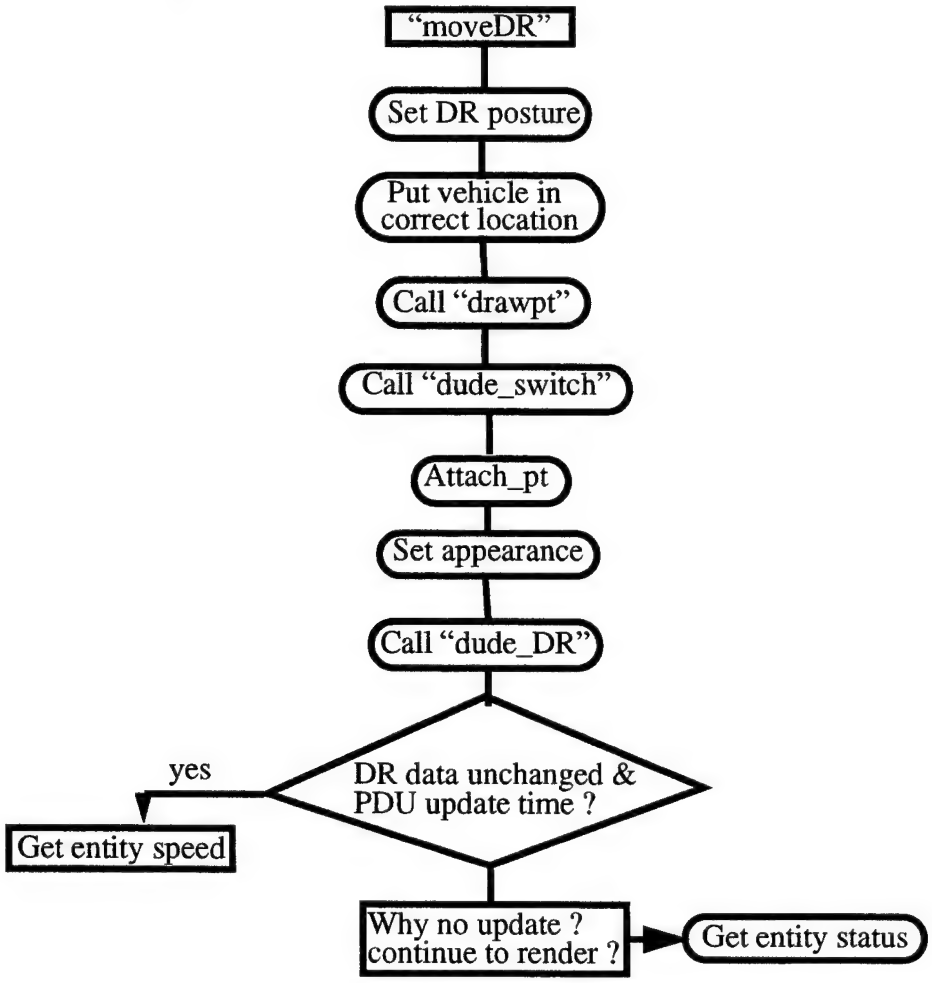
DUDECLASS-attach_pt

- Move the attached culling point to the Dude's x, y, z position
- Set the x, y, z, position of the point

f. "moveDR"

For the driven entity, the "move" routine allows updating and manipulation of the user's icon. When other remote entities are present in NPSNET-IV.8.1, their icons are updated and manipulated by movement functions which dead reckon positions based upon previous information and elapsed time. "MoveDR" provides much of the same functions to the remote entities as "move" does to the locally driven entity. In addition,

"moveDR" determines whether a network packet update, an Entity State PDU, needs to be sent out. The following flow chart shows the vital "moveDR" functions:



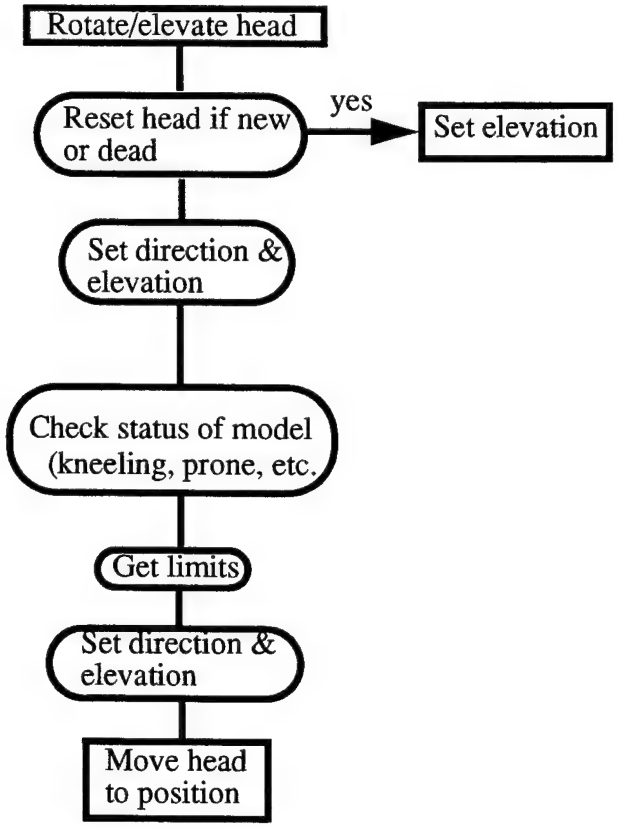
As shown above we call "drawpt" to determine if the remote Dude model should be rendered based upon the view frustum, "Dude_switch" is called to draw the correct Dude model, and "attach_pt" is called to attach the culling point to the remote Dude icon. "Dude_DR" determines the correct rendering posture and head orientation of the remote Dude icon. Also shown in the above pseudocode is the PDU update point. Incoming Entity State PDUs notify the local entity via "moveDR"of change in a remote entity's state. Even if a remote entity is not moving, a "heartbeat" PDU message is sent at regular intervals. A default standard is that a PDU must be sent at least every five seconds. If an Entity

State PDU has not been received for every remote entity in two time periods, then the local object corresponding to the remote entity is removed.

6. Special Features

From the above flow chart, it is clear that multiple types of updates are requested based upon the current status of a Dude icon. These updates make it possible to control the icon and render it in the correct posture. For example if the icon is to be rendered as kneeling and not dead we call "dude_kneel". If he is to be rendered in a dead posture, we call "dude_dead". For each Dude function update, a new head orientation is usually required to maintain realism. While it may seem trivial, head and eye orientation do not simply change with each posture change. As such, a separate function was created to accurately articulate the head and eye point. This function, called "rot_elev_head", rotates the head to match the user's inputs. We handle looking left and right as well as up and down. The head can rotate through a maximum arc of +/-75 degrees yet the eyes are able to rotate through +/-90 degrees. In this function, the head turns until it reached 75 degrees, then the view continues to shift simulating the eyes rotating in their sockets to 90 degrees. For pitch (up/down), the head moves through a maximum arc of +/-45 degrees while the eyes move through +/-90 degrees. Additionally, if the Dude icon is prone he is only able to pitch his head forward (down) through 10 degrees. Also while prone, he can only look left and right through 45 degrees. The maximum range for head and eye rotation are based upon extensive testing to achieve the most realistic movement. These ranges may be changed in compilation to reflect an individual user's preference. The following flow chart shows the main points of the "rot_elev_head" function for the driven entity As for the remote, the head is rendered

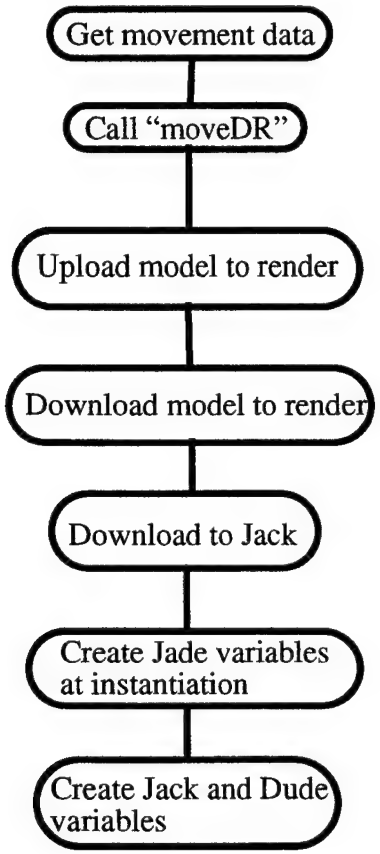
based upon DIS PDU information. The remote model head manipulation routine does not calculate elevations or alignments. .



C. JACK/DUDE (JADE) COMBINATION

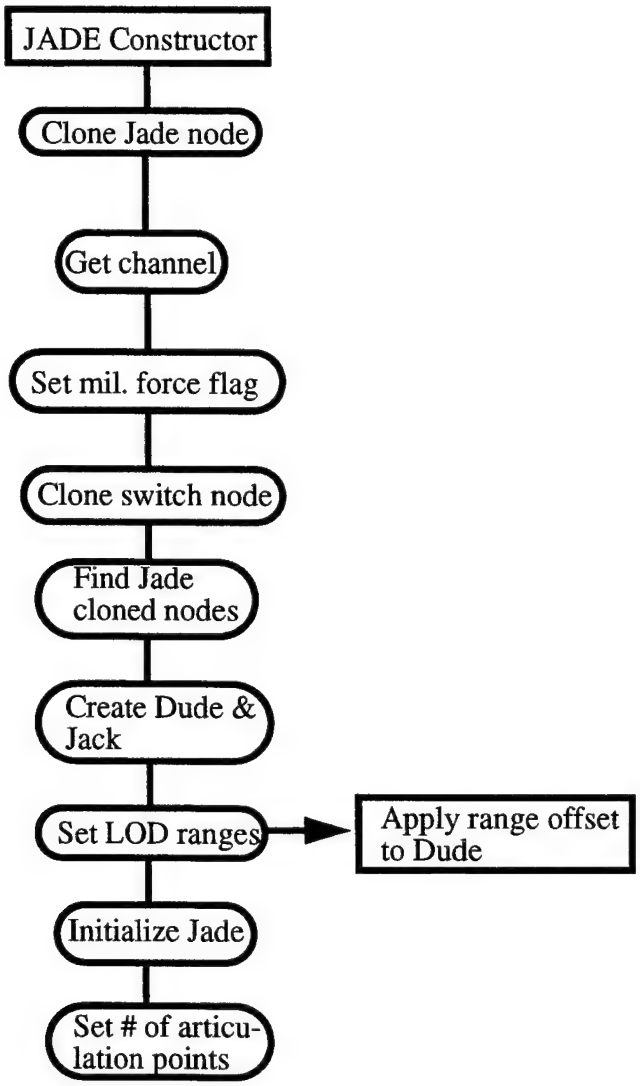
While the Dude portion of the software is designed to provide stand-alone human icon support using only Dude models, the Jade portion is designed to provide both Jack ML and Dude human icon support. For this purpose, we developed the class JADECLASS. JADECLASS was made a C++ friend of both DUDECLASS and the JACK_DI_VEH class. This allows Jade access to public as well as protected functions, variables, etc. of the two classes. The goal of this design is that the JADECLASS is the controlling logic for a switch that toggles between the two separate models, Jade and Dude, when full Jack ML icon support is required. This structure relies on the node structure created in netutil.cc as was discussed above. When using Jade, if the Jack ML icon moves to the view frustum

range where it can switch to a Dude icon, the Jade software acts as the logic switch and passes the required information to render a Dude model. The converse is also true when Dude rendering ranges approach that for Jack ML. The following pseudocode shows the logic functions needed to switch models:



The JADECLASS, similar to DUDECLASS, is derived from the PERSON_VEH. Basically, the JADECLASS is a child of PERSON_VEH that is inert when Jack ML and Dude are running in their respective stand-alone modes, but activated when running Jade,

it becomes the sibling of both. The structure of the constructor is shown in the following flow chart:



Here we call the constructor function "create_entity" which clones the structure built in "netutil.cc". Next, we set up initializers for the Jade class and then name the nodes of the clone to allow multiple Jades icons independent movement. If this were not done, all Jade icons created would have identical motions based upon the first master Jade icon. After naming, we instantiate a Dude and a Jack ML model by making calls to a special constructor in each respective class. The special constructors are shortened versions of the respective stand-alone constructors. The new constructors fill in the unique model data for

its Dude/Jack model and disregard stand-alone NPSNET-IV.8.1 model tie-ins. Also we added a variable to Dude called "Jack_offset" which handles the transition distances for the LOD in the presence/absence of a Jack ML model. When running in stand-alone mode Dude has a Jack offset of 0.0 meters so the LOD switches are set at 50, 100, and 250. When Jade is activated it sets the Jack offset to 100.0 yards. This means that Jack is used for 100 yards, then Dude is used with LOD switching occurring at 150, 200, and switching off at 350. Completing the JADECLASS is the destructor. It is a short function best summarized in the following pseudocode:

JADECLASS-~JADECLASS

- Delete Jack

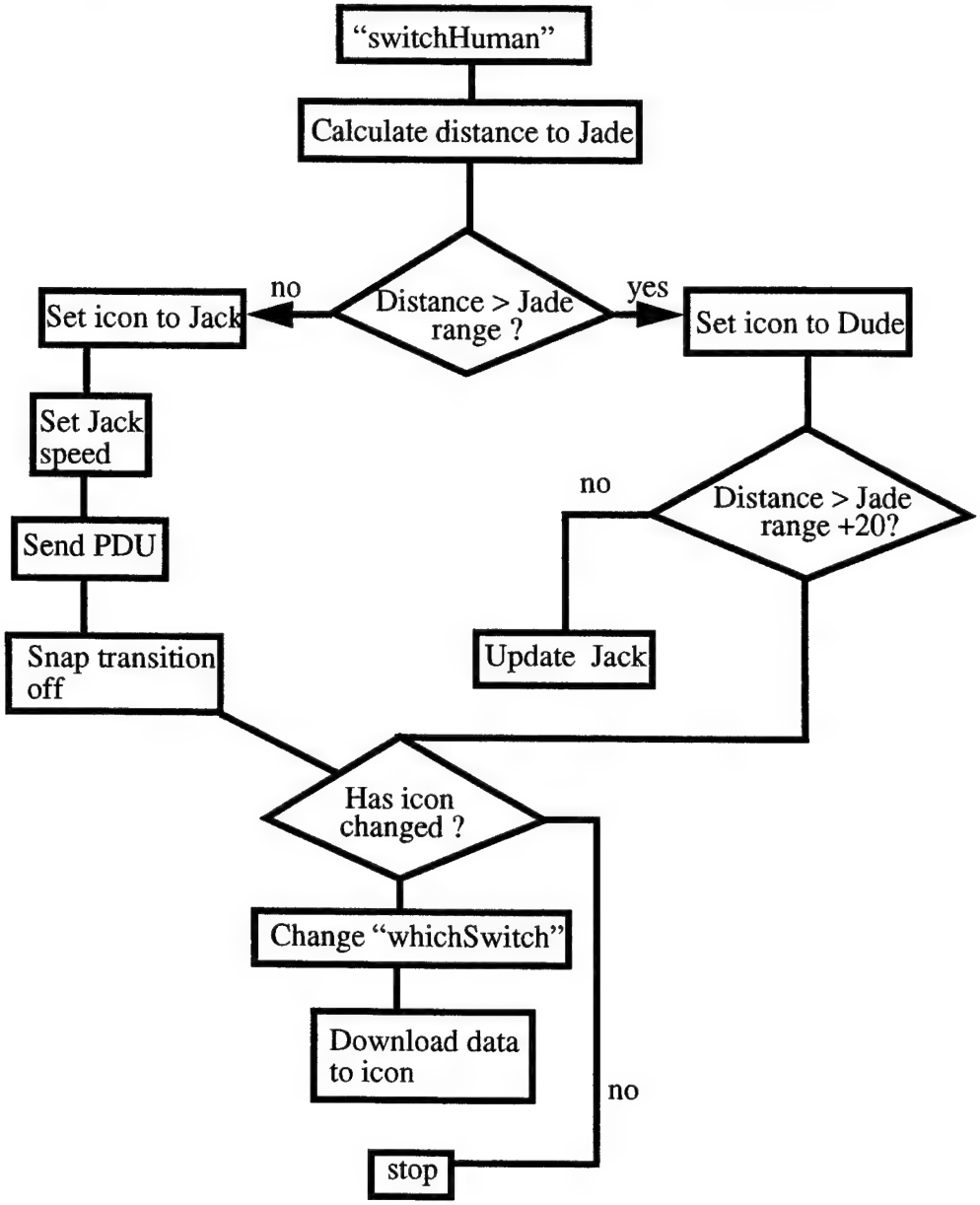
- Delete Dude

Here we simply deallocate all memory we had requested by instantiating the two classes.

1. Jade pfSwitch "whichSwitch"

The Jade pfSwitch "whichSwitch" is a controlling factor to displaying and articulating the rendered entity. This function, manipulated in the function "switchHuman," ensures there is proper data transferred to the motion algorithms of the selected icon. It

simply acts as a controlling switch through which the correct model is chosen for rendering; The following pseudocode describes some of "whichSwitch's" functions:

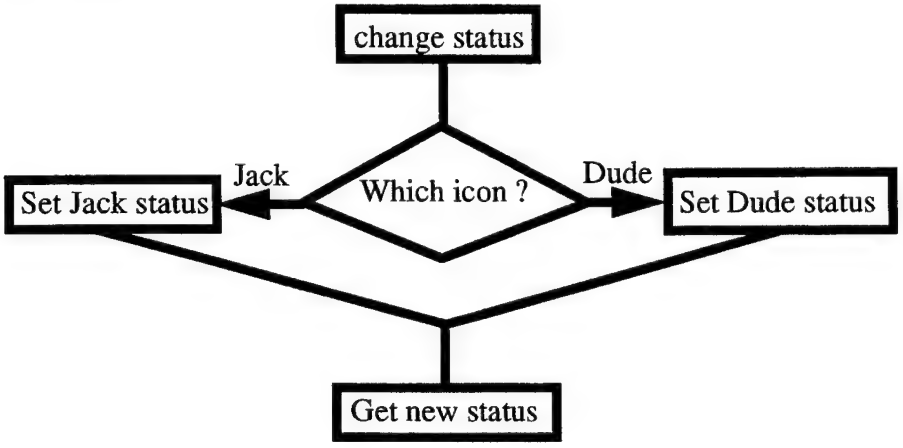


For this switch, we have a local variable set to the Jack ML icon when this function is called. We use the Euclidean distance formula determine how far away the Jade model is from the viewer's eyepoint. If the calculated distance is greater than the Jade range of 100.0 yards, the local variable is set to Dude. Also here, we check to see if we are near

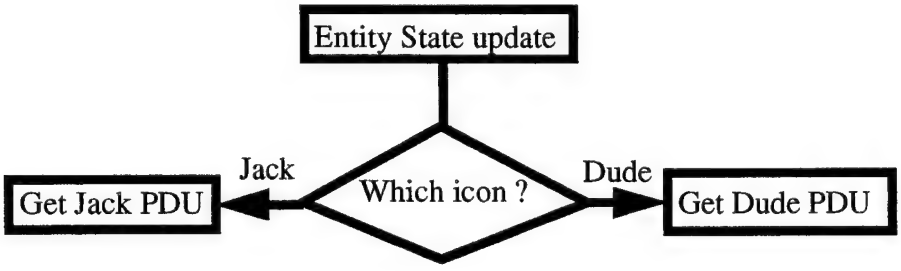
the same switching range point so as to get Jack ML updated to the same posture as Dude to make transitions look smoother. Here too, we turn off Jack's posture transition functions and simply snap Dude to position. Next we see if a model switch occurred. If it did, we give the newly selected model all of the data it needs to function properly by performing a down-load of data. If we have returned to a Jack model, we move the switch to Jack and restore Jack's intermediate posture transition algorithms.

2. Updating the Jade Model Status

After switching, we must next update the icon status. The following flow chart shows how this is performed:

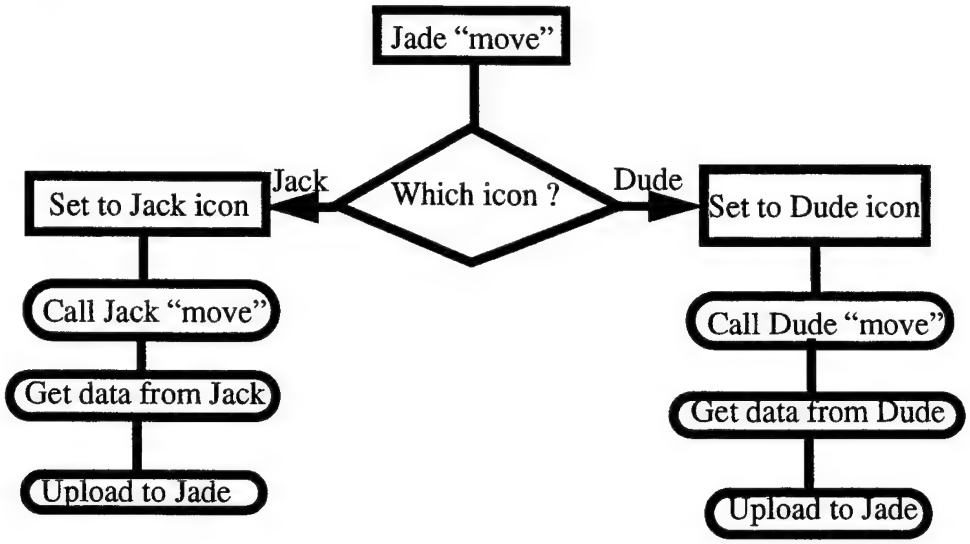


Here Jade calls the same function to update icon status in the appropriate model being displayed. We also set the velocity to 0.0 if Jade is damaged or destroyed. From here, we must receive new Entity State PDUs for networking the Jade models. This performed as in the following manner:



Here again Jade calls the same function to update icon status in the appropriate model being displayed. After having obtained entity updates, Jade is now ready to move or

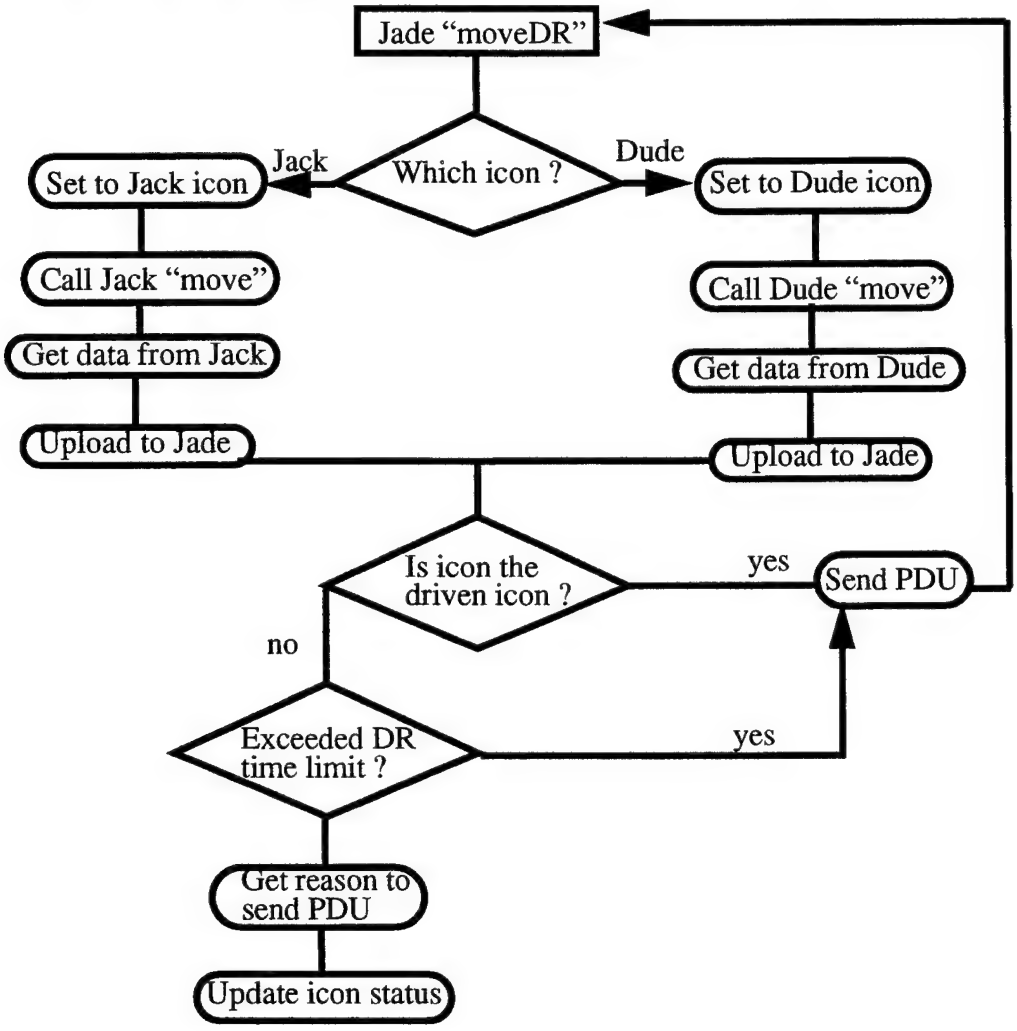
dead reckon the proper model. This is shown in the following flow chart for the Jade "move" function:



The above flow chart was designed with code reuse in mind. Simply, here we again call "which_human" to get the right model. Based upon that call, we call the appropriate class move. Finally, we upload all the necessary data from the desired model into Jade. From this we are able to move and fully support both Jack ML and Dude.

3. Jade's MoveDR

The "moveDR" function for Jade is similar in design to the above "move" function. The main goal again is code reuse, and like Dude, it is based upon elapsed time and Entity State PDUs. The design is apparent in the following flow chart for Jade "moveDR":



As one can see, this function is quiet similar to the Dude "moveDR" described in detail above. Making use of existing code, if we are not the driven vehicle, we call the appropriate "moveDR" of the model shown. Here too, one finds the functions to send Entity State PDUs to the network.

4. JADE Data Uploading and Downloading

The two critical functions below provide the smooth transfer of pertinent data between Jack ML and Dude models. The "upload" function provides the displayed model status to Jade. Similarly, the "download" function passes the required data from Jade to the next model. The following pseudocode clarifies the operations of these two functions:

JADECLASS-upload

- Upload pertinent data to Jade from correct model
- If model displayed is Dude
  - Get Dude's heading, posture, and speed
  - Get Dude status
  - Send Dude's status to Jack
  - Pass Dude head movements
- Else
  - Get Jack's heading, posture, and speed
  - Pass Jack status
  - Pass Jack head movements

Here we check to see what model is currently displayed, and get all the pertinent data from it into Jade. This function is the downloading function, described by the following pseudocode:

JADECLASS-download

- Download pertinent data from Jade to correct model
- If displayed model is Dude
  - Get Dude's heading, posture, and speed
  - Get Dude status
  - Send Dude's status to Jack
  - Pass Dude head movements
- Else
  - Get Jack's heading, posture, and speed

- Get Jack's status
- Send Jack's status to Dude
- Pass Jack's head movements

D. SUMMARY

While the addition of level of detail icons to an existing model may seem simple, their proper integration and manipulation, especially if they are articulated, is not an easy task. Extensive software additions were required to effectively add three supporting LOD icons and their associated motion algorithms to NPSNET-IV.8.1 for Jack ML support. The same extensive modifications were required to provide stand-alone Dude support for the system. Using software engineering techniques, the modifications take advantage of code reuse. Still, new functions were created that properly switch models for displaying, and transfer articulation information between the articulated icons. These new software modifications now provide NPSNET-IV.8.1 with full Jack ML LOD support, as well as fully functional, stand-alone Dude capabilities.

VI. CONCLUSIONS

A. OVERVIEW

The inclusion of level of detail models composed of fewer polygons than an existing high level model, together with view volume culling and entity range management ensure NPSNET-IV.8.1 can satisfy requirements of displaying 150 DI icons and still maintain a frame rate of 10-15 frames per second. NPSNET-IV.8.1 now has the capability of supporting the Jack ML model as well as providing stand-alone Dude model support. The inclusion of the new model types (Jade and Dude) has significantly enhanced NPSNET-IV.8.1 human interaction capabilities.

B. JACK BEFORE DUDE

As a baseline, an initial run of NPSNET-IV.8.1 using only the Jack ML icon was made. As shown in Table 6, less than 15 Jack ML icons are required to drop the frame rate below 10.0 frames per second. The large number of polygons used to create the Jack ML model, coupled with the computational expense of articulations, make it nearly impossible to maintain a real-time virtual environment composed of Jack ML icons alone.

Icon Displayed	Frames per second	Frames per second - unarticulated
12	10-12	12-15
18	6-8	8-10
50	1-2	3-4

Table 6: Jack ML Data

C. RESULTS OF DUDE

The following results were obtained for 18 Dude models on Elvis using flags -n static, and -n animations on the Fort Benning terrain database. This data was collected with 150 Dude icons in the virtual environment. When no models were displayed, our frame rates

fluctuated between 25 and 30 frames per second. Data was also collected with static objects present, but no significant change in frame rate was realized.

Icons Displayed	Frames per second	Model Resolution
24/150	12-15	6 medium, 6 med far, 12 far
42/150	10-12	12 medium, 6 med far, 24 far
18/150	12-15	6 medium far, 12 far
12/150	12-15	6 medium, 6 medium far

Table 7: 150 Dude Entities

D. RESULTS OF JADE

As a final test, Jade was used to support Jack ML again on the Fort Benning database. This data was collected with not more than six of the 18 or 36 icons in the view frustum. Of note, when all 18 icons were being rendered as Jack, the frame rates dropped by nearly half of the below values. This result is consistent with a stand-alone Jack ML exercise with all 18 Jack icons in the view frustum. A similar result occurred with 36 icons in the view frustum. Frame rates dropped to 1-2 frames per second, consistent with the Jack ML stand-alone version of 50 icons.

Icons Displayed	Frames per second	Model Resolution
36	8-10	medium
36	8-10	medium far
36	8-10	far
36	3-5	Jack
18	10-15	medium
18	12-15	medium far
18	20-30	far
18	10-12	Jack

Table 8: 18 & 36 Jade Models

E. RESULTS OF ENTITY CULLING

Entity culling is extremely effective and important to preserving frame rates. From the above model results, it is clear that by removing non-visible entities, one can preserve frame rates by discarding objects outside of the view frustum. Equally effective to ensure frame rate preservation, range management also reduced the number of managed, articulated icons in the virtual environment. While testing culling results with 150 entities does show significant frame rate preservation, entity culling will become extremely important when NPSNET-IV.8.1 users begin to exceed the normal limits of the graphics machine in large-scale exercises, i.e., exercises of 500 or more users. Nonetheless, view volume culling and range management proved highly effective in maintaining real-time performance in NPSNET-IV.8.1

F. CONCLUSIONS

From the above results, it is clear that the basic goal of this thesis was met. We succeeded in supporting 150 DI icons in NPSNET-IV.8.1. Unfortunately, it was not possible to significantly increase the number of high resolution DI icons through our efforts of LOD modeling and entity culling. From the above data, we can conclude that NPSNET-IV.8.1 will render and manage at least 150 Jade DI icons, with not more than 5-6 being displayed and articulated as the Jack ML model. The upper bound on frame rates is not the number of Jade icons present in the virtual environment since NPSNET-IV.8.1 easily supports this number when they are not all being rendered or articulated. Clearly, range management is useful to conserve frame rates by not managing entities in the distance. NPSNET-IV.8.1's frame rates suffer when the entities are being rendered and articulated in the view frustum.

G. FUTURE WORK

While the Jade/Dude software is effective in enhancing human interaction in NPSNET-IV.8.1, it is clear that this increase in interaction leads the way for more improvement in human interaction. The following is a partial list of future improvements needed to

further enhance the Jade/Dude software. Some of these improvements will become irrelevant with future increases in human icon interaction and NPSNET-IV input device. In the mean time, their contributions would do much to further enhance the new capabilities of NPSNET-IV.8.1.

1. Realistic Walking Algorithms

The current Jade/Dude walking algorithms are adequate for short term support of the Jack ML icon, however, to more closely model actual human motion, an improved walking algorithm is required for the Jade/Dude program. Smoother articulations and a closer match of speed to icon movement are required to complete the Jade/Dude software.

2. Special Activity Algorithms

Currently, activities such as climbing steps, entering building through windows, parachuting, driving, and basic human interaction other than that described in this thesis is not supported for either Jack ML, Jade, or Dude. Special scripted or interactive animations are required to fully depict these activities. The addition of special animations will tremendously increase the realism of the human interaction portion of NPSNET-IV.8.1.

Additionally, when the Dude model is damaged or destroyed, the icon is replaced with smoke and a crater, further efforts are required to override this default death script and create a special one for the human Dude ground vehicles.

3. Improved Jade Switching Algorithm

An improved switching algorithm is needed in Jade so that when switching between the models (Jack & Dude) the condition of fluttering is eliminated. This condition occurs when the viewing distance falls within a narrow range where icon movement causes it to fluctuate between 99, 100, and 101meters. A new switching algorithm is required that does not continue switching but picks one model and continues rendering it until the distance moves beyond the indecisive transition point.

LIST OF REFERENCES

[BADLER] Badler, Norman I., Phillips, Cary B., Webber, Bonnie Lynn, "Simulating Humans," Oxford University Press, New York, 1993.

[PTBARH] Barham, Paul T., Pratt, David R., Zyda, Michael J., Locke, John, Falby, John, "NPSNET-IV: A DIS Compatable, Object-Oriented, Multiprocessed Software Architecture For Virtual Environments," Research Paper, Naval Postgraduate School, Monterey, CA, 1994.

[TAFUNK] Funkhouser, Thomas A., Sequin, Carlo H., Teller, Seth J., "Management of Large Amounts of Data in Interactive Building Waklthroughs,"Computer Graphics, Proceedings of the 1992 Symposium on Interactive 3D Graphics, March 1992, pp.11.

[JPGRAN1] Granieri, John P., "Jack/TTES: A System for Production and Real-time Playback of Human Figure Motion in a DIS Environment," Research Paper, Center for Human Modeling and Simulation, University of Pennsylvania, Philadelphia, PA, August 1994.

[JPGRAN2] Granieri, John P., Badler, Norman I., "Simulating Humans in VR," Research Paper, To appear in Applications of Virtual Reality, R. Earnshaw, J. Vince, H. Jones, Editors,Academic Press, 1995.

[IDA] Institute for Defense Analyis, "SIMNET," Draft, Arlington, VA, May 1990.

[IRIS] Hartman, J., Creek, P., IRIS Performer Programming Guide, Document Number 007-1680-020, Silicon Graphics Incorporated, Mountain View, CA, November 1991.

[HLMOHN] Mohn, Howard L., "Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in MODSAF," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1994.

[DRPRATT1] Pratt, David R., "A Software Architecture for the Construction and Management of Real-Time Virtual Worlds," Dissertation, Naval Postgraduate School, Monterey, CA, June 1993.

[DRPRATT2] Pratt, David R., Barham, Paul T., Locke, John, Zyda, Michael J., et. al, "Insertion of an Articulated Human into a Networked Virtual Environment," Computer Graphics, Proceedings of the 1994 AI, Simulation and Planning in High Autonomy Systems Conference, Paper AIS-26, December 1994.

[SMPRATT] Pratt, Shirley M., Pratt, David R., Waldrop, Marianne S., Barham, Paul T., Ehlert, James F., Chrislip, Christopher A., "Humans in Large-Scale, Real-Time, Networked Virtual Environments," Submitted to Presence, May 1995.

[THORP] Thorpe, Jack A., "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," Proceedings of the 9th Interservice/Industry Training System Conference, Nov-Dec 1987.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-1501	2
3.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
4.	Dr David R. Pratt, Code CS/PR Computer Science Department Naval Postgraduate School Monterey, CA 93943	3
5.	Dr Michael J. Zyda, Code CS/ZK Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
6.	Shirley M. Pratt, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
7.	LT Christopher A. Chrislip 618 W. 9th St. Newton, NC 28658	2
8.	LT James F. Ehlert, Jr. 95-270 Waikalani Dr # C-301 Mililani, HI 96789	2